CURSO PRÁTICO 49 DE PROGRAMAÇÃO DE COMPUTADORES

# INFO

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

Cz$ 35,00

# INPUT

Vol. 4 N.º 49

## NESTE NÚMERO


**PROGRAMAÇÃO DE JOGOS**

### O JOGO DA VIDA

As regras. Disposição inicial das colônias. Criação da colônia. O programa em código ..... 961

**PERIFÉRICOS**

### TABLETES GRÁFICOS

Digitalização e mapeamento. Geração de sinais. Sensores de superfície. Software..... 964

**CÓDIGO DE MÁQUINA**

### AVALANCHE: A ROTINA PRINCIPAL

Tarefas da rotina principal. Origem. Acertos iniciais. Dando a partida ..... 969

**PROGRAMAÇÃO BASIC**

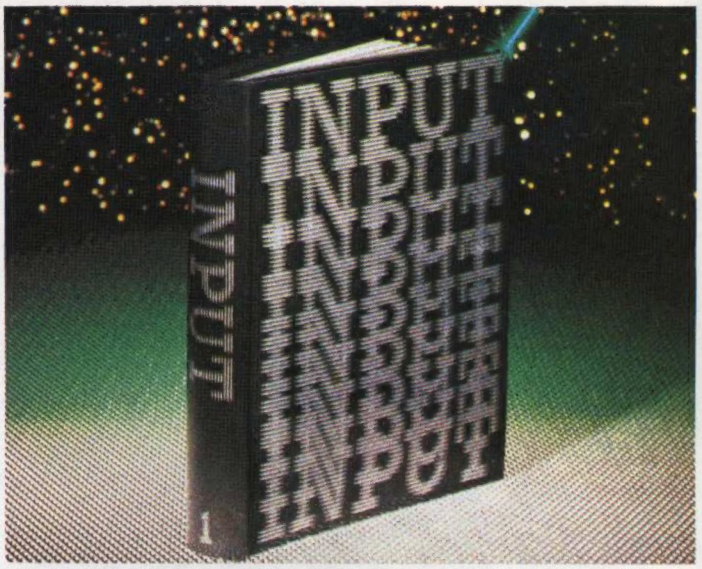
### ROTINAS EM CÓDIGO DE MÁQUINA (1)

O uso de rotinas em código dentro de programas Basic. Os comandos **USR** e **DEFUSR** ..... 972

**PROGRAMAÇÃO DE JOGOS**

### A ARANHA MARCIANA (2)

O jogo completo. Laço principal. Rotinas de animação das figuras ..... 976


## PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

## COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: 1. PESSOALMENTE — Por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em **São Paulo**, os endereços são: rua Brigadeiro Tobias, 773, Centro; avenida Industrial, 117, Santo André; e no **Rio de Janeiro**: avenida Mem de Sá, 191/193, Centro. 2. POR CARTA — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidora Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132, Jardim Teresa — CEP 06000 — Osasco — SP. Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na agência do Correio. 3. POR TELEX — Utilize o n.º (011) 33 670 DNAP.

Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações, Lda. — Qta. Pau Varais, Azinhaga de Fetais — 2 685, Camarate — Lisboa; Apartado 57 — Telex 43 069 JARLIS P.

**Atenção:** Após seis meses do encerramento da coleção, os pedidos serão atendidos dependendo da disponibilidade do estoque.

**Obs.:** Quando pedir livros, mencione sempre título e/ou autor da obra, além do número da edição.

## COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.


**Editor**
VICTOR CIVITA

**REDAÇÃO**
**Diretor Editorial:** Carmo Chagas

**Editores Executivos:** Antonio José Filho, Berta Sztark Amar

**Editor Chefe:** Paulo de Almeida
**Editor de Texto:** Cláudio A. V. Cavalcanti
**Chefe de Arte:** Carlos Luiz Batista
**Assistentes de Arte:** Dagmar Bastos Sampaio, Grace Alonso Arruda, Monica Lenardon Corradi
**Secretária de Redação/ Coordenadora:** Stefania Crema
**Secretários de Redação:** Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz

**COLABORADORES**

**Consultor Editorial Responsável:** Dr. Renato M. E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
**Execução Editorial:** DATAQUEST Assessoria em Informática Ltda., Campinas, SP
**Tradução, adaptação, programação e redação:** Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Marcos Huascar Velasco, Raul Neder Porrelli, Ricardo J. P. de Aquino Pereira
**Coordenação Geral:** Rejane Felizatti Sabbatini
**Editora de Texto:** Ana Lúcia B. de Lucena

**COMERCIAL**
**Diretor Comercial:** Roberto Martins Silveira
**Gerente Comercial:** Flávio Maculan
**Gerente de Circulação:** Denise Maria Mozol

**PRODUÇÃO**
**Gerente de Produção:** João Stungis
**Coordenador de Impressão:** Atílio Roberto Bonon

**Preparador de Texto/Coordenador:** Eliel Silveira Cunha
**Preparadores de Texto:** Alzira Moreira Braz, Ana Maria Dilguerian, Levon Yacubian, Luciano Tasca, Maria Teresa Galluzzi, Maria Teresa Martins Lopes, Paulo Felipe Mendrone
**Revisor/Coordenador:** José Maria de Assis
**Revisoras:** Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria de Fátima Cardoso, Nair Lucia de Britto
**Paste-up:** Anastase Potaris, Balduino F. Leite, Edson Donato



© Marshall Cavendish Limited 1984/85.
© Editora Nova Cultural Ltda., São Paulo, Brasil, 1986.


Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, n.º 2000 - 3º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

# O JOGO DA VIDA

- AS REGRAS DA VIDA
- A DISPOSIÇÃO INICIAL DA COLÔNIA
- COMO CRIAR UMA COLÔNIA
- O PROGRAMA EM CÓDIGO

**Programe seu micro para simular a eterna luta dos seres unicelulares pela vida. Depois, desafie um parceiro para jogar. O arranjo inicial da colônia de bactérias definirá o vencedor.**

Neste fascinante jogo, o vídeo representa um mundo bidimensional, onde células podem viver, multiplicar-se e morrer. O *Jogo da Vida*, como é denominado, foi inventado por um cientista inglês há alguns anos, para ser jogado em um tabuleiro como o de xadrez. Em sua versão computadorizada, bastante popular entre os usuários de microcomputadores, a tela é dividida em um padrão quadriculado (invisível para o jogador). Cada quadradinho pode abrigar uma célula — uma bactéria, por exemplo. Essa célula terá, de acordo com sua disposição no diagrama, até um máximo de oito vizinhos.

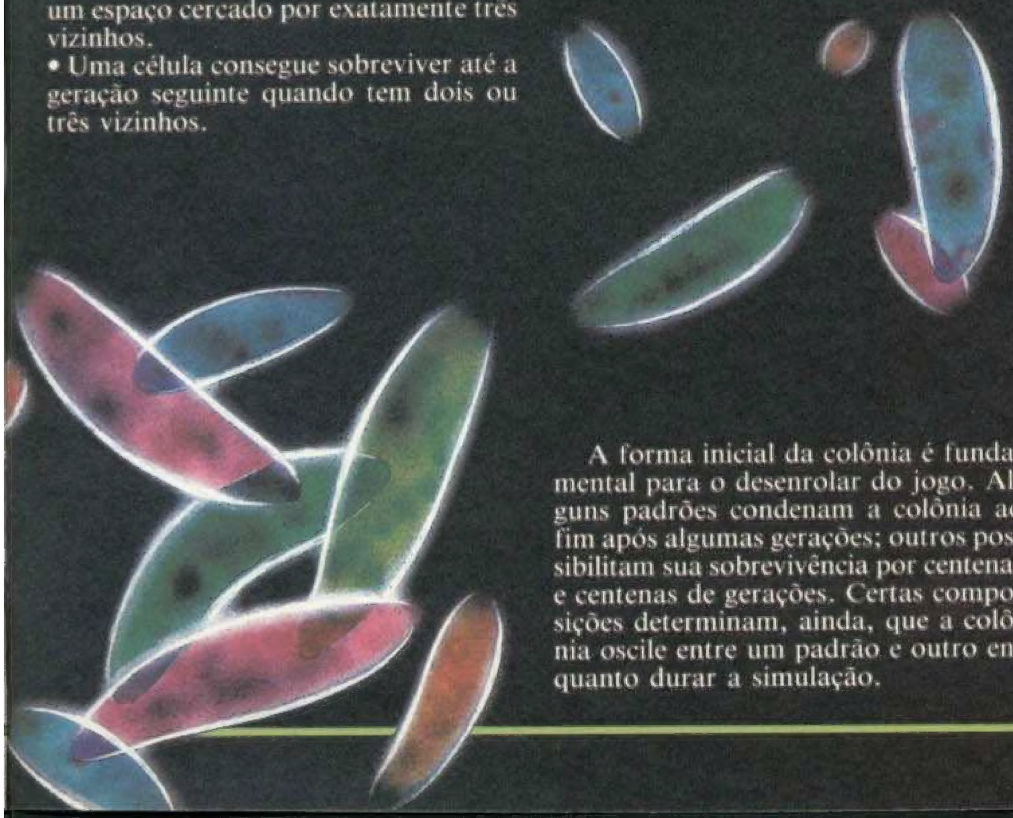
As regras do *Jogo da Vida* determinam quando uma célula deve sobreviver ou morrer, e, também, quando uma nova célula deve nascer. São elas:

- Uma célula nasce sempre que existe um espaço cercado por exatamente três vizinhos.
- Uma célula consegue sobreviver até a geração seguinte quando tem dois ou três vizinhos.
- As células que não se enquadrarem nas situações anteriores morrem. Em um espaço com mais de três vizinhos, por exemplo, supõe-se que faltará alimento ou oxigênio para todas as células.

Baseado nessas regras, o programa determina o futuro de cada quadrado na tela e mostra como a colônia inicial, montada pelo jogador, se desenvolve de geração para geração. Cada geração corresponde a um ciclo completo de cálculos para todo o quadriculado.

Dependendo do tamanho da grade — ou seja, do número de quadradinhos —, o cálculo de uma geração será muito demorado se se utilizar um programa escrito em linguagem BASIC. Por isso, montamos o programa em linguagem de máquina, o que lhe permitirá observar uma geração da colônia a cada segundo, aproximadamente.

Como as gerações são exibidas em cores diferentes, o efeito visual é muito interessante. Você verá, na tela, como as colônias se espalham, mudam, fragmentam-se em unidades menores, morrem ou rejuvenescem.



A forma inicial da colônia é fundamental para o desenrolar do jogo. Alguns padrões condenam a colônia ao fim após algumas gerações; outros possibilitam sua sobrevivência por centenas e centenas de gerações. Certas composições determinam, ainda, que a colônia oscile entre um padrão e outro enquanto durar a simulação.

Cabe ao jogador estabelecer o padrão inicial, entrando as posições das primeiras células. Um dos objetivos do jogo consiste em criar uma colônia que dure o maior número possível de gerações — o programa informa quantas gerações se passaram. Entretanto, às vezes, vale a pena competir simplesmente para ver quem compõe a colônia com efeito mais interessante. Existem alguns padrões, por exemplo, que se deslocam em uma direção, recriando sua forma original a cada quatro ou cinco gerações, com um efeito adicional: a eliminação de todas as células que vão sendo encontradas no caminho.

Você poderá aproveitar tal efeito para desenvolver um jogo em que dois oponentes montam formas "devoradoras", ganhando aquele que destruir mais depressa a colônia do outro.

## O PROGRAMA

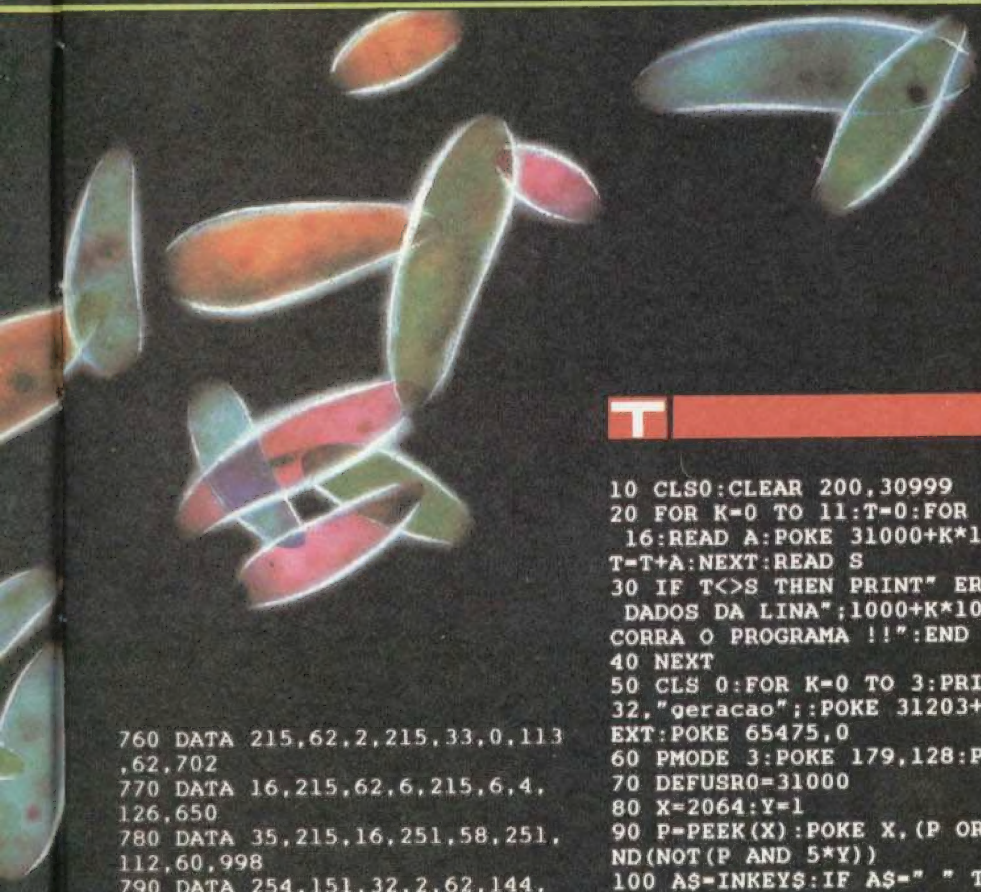
Não entraremos em detalhe quanto ao funcionamento do programa em linguagem de máquina. Este é carregado e acionado por meio de um programa em linguagem BASIC que utiliza os comandos **POKE** e **USR**. Os códigos decimais correspondentes ao programa estão armazenados nas instruções **DATA** que começam na linha 500 (no Spectrum) ou 1000 (no TRS-Color). A parte do programa escrito em BASIC simplesmente lê esses códigos, colocando-os numa parte protegida da memória. Além disso, ela é responsável pela criação da tela e pela entrada da colônia inicial.

Este jogo é um interessante exemplo do uso de programas em código de máquina a partir de um programa em BASIC. Apresentamos aqui as versões para os micros da linha Spectrum e TRS-Color. Os programas para outras máquinas serão dados posteriormente.

```
 5 CLEAR 28671: FOR N=USR "A
 " TO USR "A"+7: READ A:
 POKE N,A: NEXT N
 6 DATA 0,24,60,102,102,60,24
 ,0
 7 GOSUB 200
10 POKE 23658,8: BRIGHT 1:
```

```
BORDER 0: INK 6: PAPER 0: CLS

20 FOR N=0 TO 21: PRINT AT N,
0; PAPER 1;"          ": NEXT N
30 PLOT 63,0: DRAW 0,175:
DRAW 192,0: DRAW 0,-175: DRAW
-255,0: DRAW 0,175: DRAW 63,0
40 PRINT AT 2,1; PAPER 2; INK
7;" VIDA "; PAPER 0; INK 6;AT
5,1;" GER  ";AT 6,1;" 0000 "
70 RAND USR 28672: LET X=20:
LET Y=10
80 PRINT AT Y,X; OVER 1;"■":
FOR N=1 TO 10: NEXT N: PRINT
AT Y,X; OVER 1;" "
90 LET A$=INKEY$: IF A$=""
THEN  GOTO 80
92 IF A$="Q" THEN  GOTO 110
94 IF CODE A$=13 THEN  PRINT
AT Y,X;CHR$ 144: POKE 30000+(
(Y-1)*23)+(X-8),144: GOTO 80
95 IF A$="M" THEN  PRINT AT Y
,X;" ": POKE 30000+((Y-1)*23)
+(X-8),32: GOTO 80
100 LET X=X-(A$="5")*(X>8)+(A$
="8")*(X<30): LET Y=Y-(A$="7")
*(Y>1)+(A$="6")*(Y<20): GOTO
80
110 OVER 0: RAND USR 28711
120 PRINT AT 21,1; FLASH 1;"QU
ALQUER TECLA PARA RECOMECAR":
FOR N=1 TO 200: NEXT N
130 LET A$=INKEY$: IF A$=""
THEN  GOTO 130
140 GOTO 10
200 LET L=500: RESTORE L: FOR
N=28672 TO 28951 STEP 8
210 LET T=0: FOR D=0 TO 7
220 READ A: POKE N+D,A: LET T=
T+A: NEXT D: READ A: IF A<>T
THEN  PRINT  FLASH 1;"ERRO DE
 DADOS NA LINHA ";L: STOP
230 LET L=L+10
240 NEXT N
250 RETURN
500 DATA 33,25,117,1,211,3,62,
32,484
510 DATA 119,35,13,32,249,5,
242,6,701
520 DATA 112,33,48,48,34,0,113
,34,422
530 DATA 2,113,33,48,117,34,
252,112,711
540 DATA 33,248,118,34,254,112
,201,243,1243
550 DATA 42,252,112,6,1,197,88
,6,704
560 DATA 8,80,62,22,215,123,
215,122,847
570 DATA 215,126,35,254,32,40,
10,214,926
580 DATA 142,245,62,16,215,241
,215,62,1198
590 DATA 144,215,4,120,254,31,
32,233,1033
600 DATA 193,4,120,254,21,32,
214,42,880
610 DATA 252,112,237,91,254,
112,229,213,1500
620 DATA 1,200,1,197,221,33,4,
113,770
630 DATA 1,0,7,213,221,94,0,
221,757
640 DATA 86,1,229,25,235,225,
26,254,1081
650 DATA 32,209,40,1,12,221,35
,221,771
660 DATA 35,5,242,107,112,126,
254,144,1025
670 DATA 121,56,18,254,2,40,4,
254,749
680 DATA 3,32,14,126,254,32,32
,11,504
690 DATA 58,251,112,24,6,254,3
,40,748
700 DATA 242,62,32,18,35,19,
193,13,614
710 DATA 32,185,5,242,99,112,
209,225,1109
720 DATA 237,83,252,112,34,254
,112,33,1117
730 DATA 4,113,43,126,60,254,
58,40,698
740 DATA 4,119,195,203,112,62,
48,119,862
750 DATA 195,186,112,62,22,215
,62,6,860
```



```
760 DATA 215,62,2,215,33,0,113
,62,702
770 DATA 16,215,62,6,215,6,4,
126,650
780 DATA 35,215,16,251,58,251,
112,60,998
790 DATA 254,151,32,2,62,144,
50,251,946
800 DATA 112,62,127,219,254,31
,218,40,1063
810 DATA 112,251,201,149,248,
118,48,117,1244
820 DATA 48,49,54,49,233,255,
234,255,1177
830 DATA 1,0,24,0,23,0,22,0,70
840 DATA 255,255,232,255,0,0,0
,0,997
```

Para entrar a colônia inicial, mova o cursor usando as teclas de controle (flechas). Pressione <**ENTER**> para criar uma célula na posição desejada, **M** para matar uma célula, e **Q** para terminar a entrada e iniciar o jogo.

```
10 CLS0:CLEAR 200,30999
20 FOR K=0 TO 11:T=0:FOR J=0 TO
 16:READ A:POKE 31000+K*17+J,A:
T=T+A:NEXT:READ S
30 IF T<>S THEN PRINT" ERRO NOS
 DADOS DA LINA";1000+K*10;"NAO
CORRA O PROGRAMA !!":END
40 NEXT
50 CLS 0:FOR K=0 TO 3:PRINT @K*
32,"geracao";:POKE 31203+K,48:N
EXT:POKE 65475,0
60 PMODE 3:POKE 179,128:PCLS
70 DEFUSR0=31000
80 X=2064:Y=1
90 P=PEEK(X):POKE X,(P OR 5*Y)A
ND(NOT(P AND 5*Y))
100 A$=INKEY$:IF A$=" " THEN 90
110 POKE X,P
120 IF A$="^" AND X>1183 THEN X
=X-32
130 IF A$=CHR$(10) AND X<3040 T
HEN X=X+32
140 IF A$=CHR$(8) AND (X>1152 O
R(X=1151 AND Y=1)) THEN Y=Y+1:I
F Y>2 THEN Y=1:X=X-1
150 IF A$=CHR$(9) AND (X<3071 O
R(X=3071 AND Y=2)) THEN Y=Y-1:I
F Y<1 THEN Y=2:X=X+1
160 IF A$=CHR$(13) THEN 180
170 GOTO 90
180 H=USR0(0)
190 IF INKEY$<>"Q" THEN 180
200 GOTO 50
1000 DATA 182,121,226,139,16,18
3,121,226,142,121,227,108,132,1
66,128,129,58,2425
```

Após 115 gerações, seis colônias sobrevivem num padrão estável (TRS-Color).

**É possível usar um sintetizador de voz para animar o jogo?**

Você pode tornar o jogo ainda mais interessante programando seu micro para anunciar mensagens ou descrever o que se passa na tela.

Recorra ao manual do seu sintetizador para ver como introduzir as instruções que tornarão possível fazer a máquina falar.

Já publicamos em INPUT um artigo que examina em detalhe o uso de sintetizadores de voz. Consulte-o.

```
1010 DATA 37,9,134,48,167,31,14
0,121,231,37,239,198,4,206,4,11
,142,1759
1020 DATA 121,231,166,130,167,1
92,140,121,227,38,247,51,200,28
,90,38,238,2425
1030 DATA 142,13,0,204,0,0,237,
129,140,28,128,37,249,142,4,128
,206,1787
1040 DATA 13,64,166,128,133,10,
39,2,141,47,51,65,133,5,39,2,14
1,1179
1050 DATA 39,51,65,140,12,0,37,
233,142,4,128,206,13,64,166,132
,52,1484
1060 DATA 2,230,192,134,10,141,
52,230,192,134,5,141,46,53,2,16
7,128,1859
1070 DATA 140,12,0,37,231,57,52
,2,108,200,192,108,200,64,31,48
,196,1678
1080 DATA 63,39,8,108,200,191,1
08,95,108,200,63,193,63,39,8,10
8,200,1794
1090 DATA 193,108,65,108,200,65
,53,130,165,98,39,19,193,2,39,3
2,193,1702
1100 DATA 3,39,28,230,98,67,52,
2,228,224,231,98,32,17,193,3,38
,1583
1110 DATA 13,230,98,196,143,250
,121,226,52,2,234,224,231,98,57
,0,0,2175
```

Para entrar a colônia inicial, movimente o cursor utilizando as teclas de controle (flechas). Pressione a barra de espaço para criar ou eliminar uma célula na posição desejada, e a tecla <**ENTER**> para terminar a entrada e iniciar o jogo.

Um detalhe importante: este programa não funcionará em um TRS-Color ou compatível que tenha um acionador de disquetes conectados, pois usa uma área de memória reservada.

# TABLETES GRÁFICOS

**Desenhar na tela do computador usando centenas de comandos em BASIC não é um procedimento razoável. Com o auxílio de um tablete gráfico, você trabalhará tão facilmente quanto com lápis e papel.**

Uma área de aplicação em que os computadores realmente excederam todas as expectativas é a do desenho auxiliado por computador, ou CAD (do inglês *Computer Aided Design*). Máquinas de todos os tamanhos, dos micros domésticos aos computadores de grande porte (*mainframes*), têm sido usadas nas mais diversas áreas do desenho e do projeto gráfico. Desenhistas de moda, por exemplo, ou empresas de construção civil, já empregam programas de CAD, e não apenas em projetos complexos.

Muitos micros domésticos foram planejados de modo a ter boa capacidade gráfica simplesmente para a implementação de jogos do tipo videogame. Desse ponto de partida, porém, desenvolveram-se programas de desenho gráfico com recursos espetaculares, sobretudo se levarmos em conta o tamanho e custo dos micros a que se destinam.

Um dos periféricos mais úteis para a exploração da capacidade gráfica dos computadores é o tablete gráfico, também chamado de *mesa digitalizadora*. O tipo mais comum permite que o usuário desenhe sobre uma superfície dura (o tablete), com o auxílio de uma caneta. O movimento da caneta no tablete é duplicado na tela do computador por meio de uma criativa combinação de hardware e software.

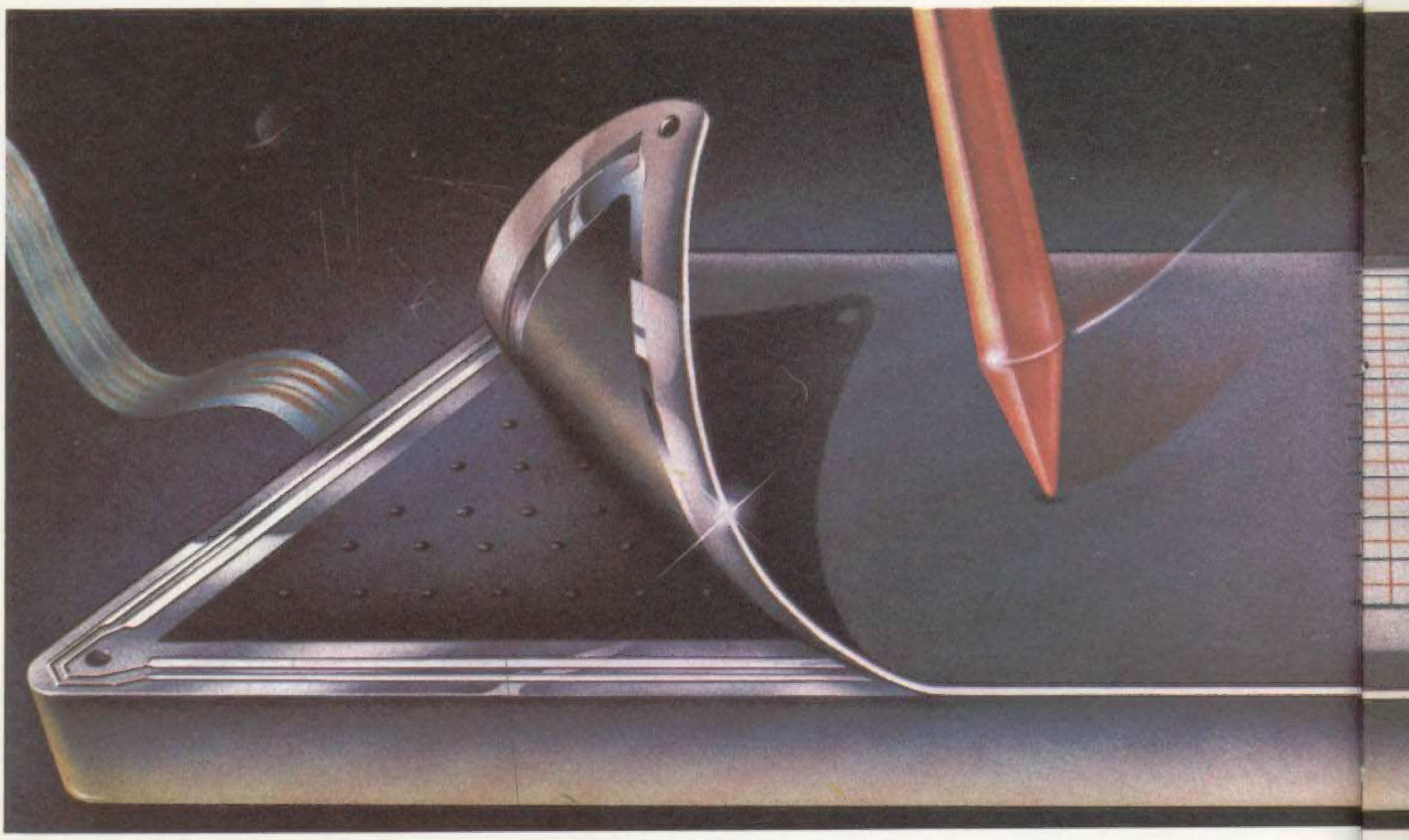
Para obter resultados equivalentes, sem empregar um tablete gráfico, seria preciso recorrer a um considerável volume de programação. Mesmo assim, não se conseguiria o mesmo grau de flexibilidade, sobretudo na alteração ou eliminação de partes do desenho.

Até os tabletes mais simples permitem o desenho de linhas sobre a tela. Dependendo do programa com o qual são usados, podem-se "pintar" áreas inteiras com uma certa cor, entre as disponíveis em seu micro. Os programas possibilitam ainda a mudança instantânea de cores no desenho, sob o comando de algumas teclas ou por seleção feita no próprio tablete. Efeitos de "pinceladas" de diferentes larguras também podem ser simulados pelo programa, em combinação com o tablete.

## TIPOS DE TABLETE

Existem diversos tipos de tablete para microcomputadores pessoais. O mais conhecido no Brasil é o KoalaPad, destinado aos modelos da linha Apple. De baixo custo em relação aos demais, apresenta algumas limitações, como a baixa resolução gráfica.

Muitos outros modelos de tablete, como o DigiPad, o GrafPad (para o Spec-



- AS VANTAGENS DOS TABLETES GRÁFICOS
- OS DIFERENTES TIPOS
- DIGITALIZAÇÃO E MAPEAMENTO
- GERAÇÃO DOS SINAIS
- SENSORES DE SUPERFÍCIE
- SOFTWARE
- COMO USAR UM TABLETE
- GUARDE SUAS OBRAS DE ARTE


trum) e o BitPad One (para diversos tipos de micro), podem ser encontrados no exterior. No Brasil, são mais comuns os tabletes digitalizadores para uso profissional, como os da marca Digigraf, bastante caros.

Os tabletes gráficos são vendidos em tamanhos que variam conforme a área de desenho disponível. Normalmente, suas medidas são especificadas segundo as dimensões padronizadas das folhas de papel (padrão DIN), tal como A3, A4, A6 etc. Medem, assim, de 100 × 100 mm (os menores, como o KoalaPad) a 250 × 300 mm (tabletes médios, usados com micros pessoais). Os modelos utilizados com mesas digitalizadoras de grande porte, para aplicações em engenharia, chegam a ter 1 000 × 750 mm. O tamanho, evidentemente, afeta o grau de precisão que se consegue atingir ao desenhar na superfície do tablete.

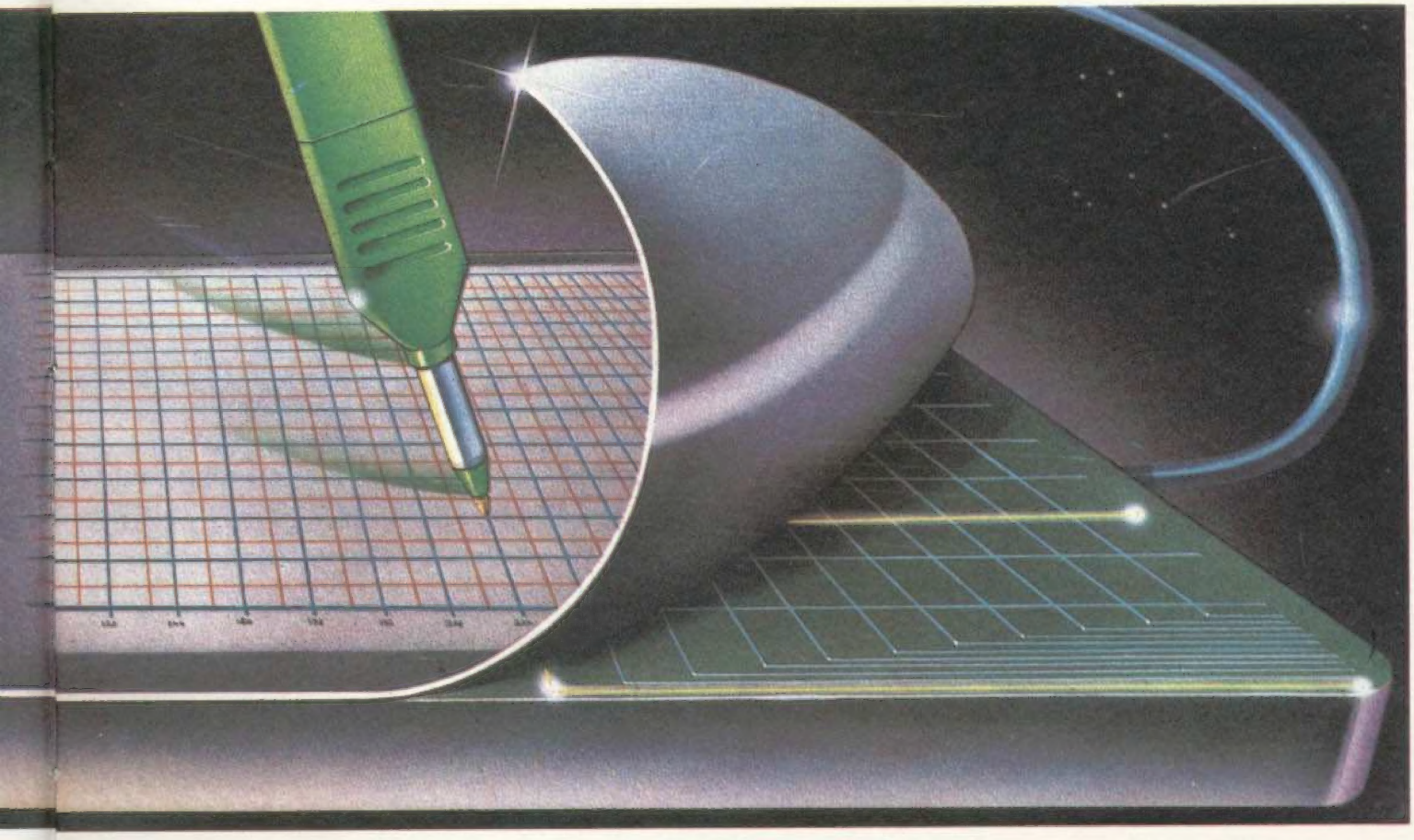
Os tabletes variam também conforme o cursor. Este pode ser um ponteiro ou uma caneta, preso ou não ao tablete por um fio, ou, então, uma "mira" formada por uma lente de vidro ou plástico, com uma cruz ao centro.

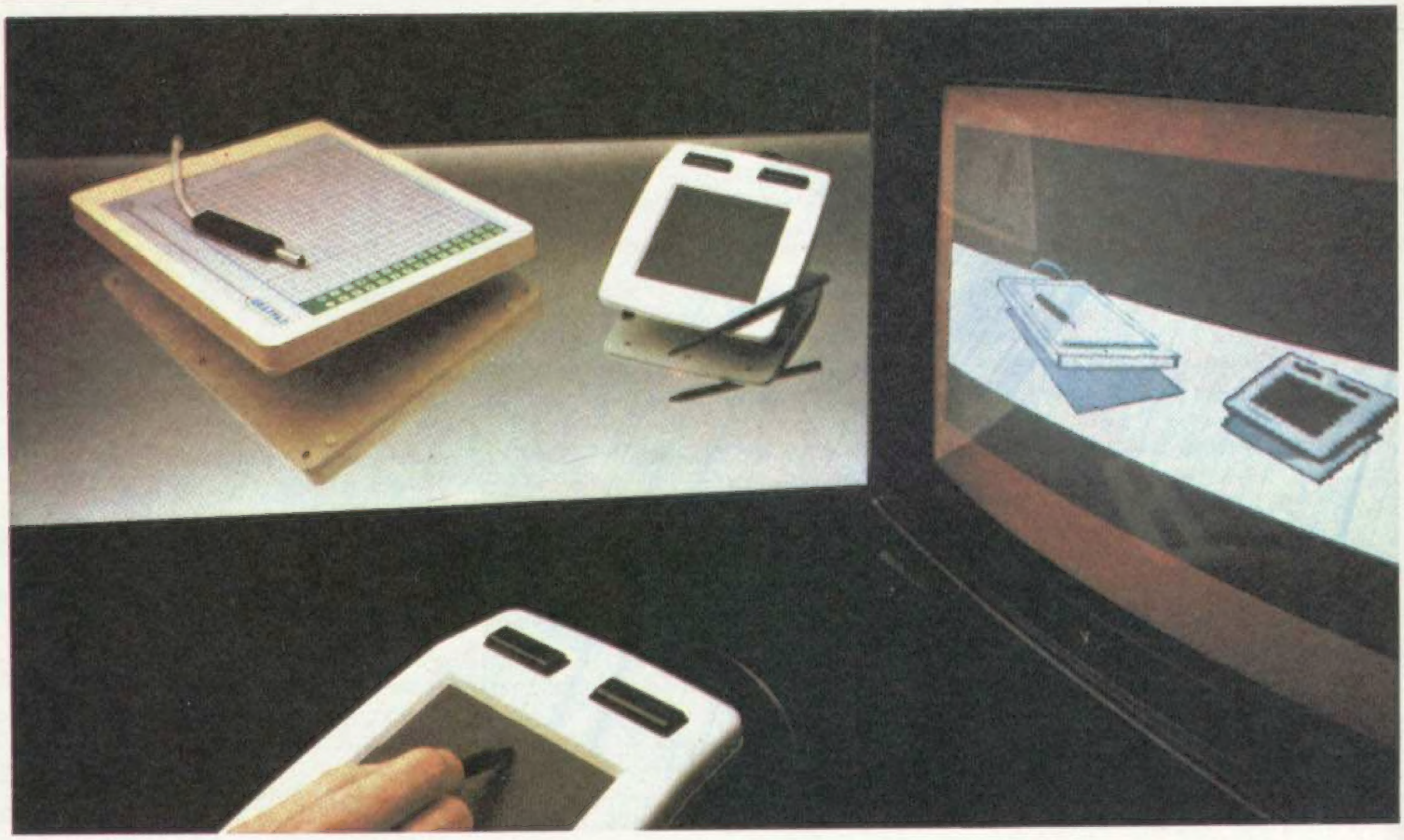
Existem ainda alguns tipos de digitalizadores gráficos que funcionam de modo diferente dos tabletes: são os *pantógrafos*, ou *traçadores de régua*. Esses dispositivos utilizam um braço articulado que se move em duas direções, X e Y. Na ponta do braço, prende-se uma caneta ou um lápis. O sistema pode ser mais barato do que um digitalizador de desempenho médio, mas, em contrapartida, sua precisão é menor. Além disso, não é fácil desenhar à mão livre com o pantógrafo. Em algumas situações, porém, esse dispositivo mostra-se de bastante utilidade — por exemplo, quando as coordenadas de pontos isolados precisam ser lidas rapidamente pelo computador.

## COMO FUNCIONA

O tablete gráfico emprega o princípio da digitalização de curvas, amplamente usado em informática, já que muitas aplicações computacionais consistem na transformação adequada de um fenômeno analógico para uma seqüência de números binários (digitais).

Um sinal analógico varia continuamente e pode assumir qualquer valor fracionário, dentro de certos limites mínimo e máximo. Um exemplo de medidor analógico é o indicador de velocidade de um automóvel. Mas, como sabemos, o computador só é capaz de processar sinais digitais, ou seja, representados por meio dos números binários 0





e 1. A digitalização é justamente o processo de transformação de um sinal analógico em um sinal digital que o computador possa "entender" e armazenar.

O tablete gráfico, ou mesa digitalizadora, transforma desenhos em números (as coordenadas dos pontinhos que constituem as linhas do desenho). O computador, por sua vez, volta a transformar os números em desenhos, exibindo-os na tela com o máximo de fidelidade ao original. Sem a tradução de uma figura em números, o computador não poderia "entender" o desenho.

O processo de digitalização consiste em dividir uma curva no maior número possível de segmentos iguais, ou uma área no maior número possível de quadradinhos de igual tamanho. Quanto maior o número de partes, maior o detalhamento do desenho a ser entrado no computador. Contornos de figuras ou retas inclinadas em ziguezague são típicos de desenhos de baixa resolução.

O número e o tamanho dos quadrados de divisão da imagem dependem do tipo do tablete gráfico, assim como do computador. Em sistemas profissionais de alta resolução, o tamanho de um pixel chega a um centésimo de milímetro, e os contornos das figuras produzidas têm uma aparência contínua (como os desenhos animados gerados por computador). Os tabletes gráficos para micros não têm uma resolução tão alta, evidentemente, mas um décimo de milímetro já é suficiente para a produção de trabalhos de boa qualidade.

A área do tablete (bem como a do vídeo) é dividida em um quadriculado imaginário, cujo número de linhas e colunas varia de acordo com o computador empregado ou, ainda, com o grau de resolução gráfica (baixa, média e alta, nos micros das linhas Apple, MSX e TRS-Color) selecionado pelo programa ou pelo usuário.

É importante lembrar que a resolução máxima do tablete pode ser diferente da resolução gráfica da tela. Não tem sentido, por exemplo, utilizar um tablete com resolução de 100 000 pontos na área total, se a tela só representa 45 000 pontos. Estaremos desperdiçando recursos, a um certo preço, pois o computador não seria capaz de utilizar tanta informação gráfica. A situação contrária também ocorre — ou seja, pode-se utilizar um computador de alta resolução gráfica com um tablete de baixa resolução.

Muitos tabletes digitalizadores têm, no entanto, superfícies quadriculadas que permitem estabelecer uma correspondência com a representação na tela. As linhas e colunas podem ser numeradas, como se costuma fazer com a grade de referência de uma mapa.

Nesse tipo de grade, cada quadradinho tem duas coordenadas — o número da linha e o número da coluna — que correspondem, aproximadamente, às coordenadas X e Y do ponto do centro do quadradinho. Em um sistema onde o ponto de origem estivesse no canto inferior esquerdo do tablete, o quadradinho de número 10/23, por exemplo, estaria a uma distância de dez quadradinhos da esquerda, na horizontal, e a 23 quadradinhos de distância da borda inferior, na vertical. Isso significa, portanto, que um desenho — uma reta, por exemplo — pode ser convertido em uma série de números. Estes correspondem às coordenadas dos quadradinhos que a reta vai cruzar ao ser traçada sobre a superfície do tablete.

## A GERAÇÃO DO SINAL

Existem diversas maneiras de produzir e enviar ao computador os pares de números correspondentes aos pontos da superfície do tablete.

Alguns tipos de mesa digitalizadora requerem uma caneta ou um apontador especial e dependem do contato direto da ponta desse dispositivo contra uma parte ativa da superfície; já outros exigem simplesmente o posicionamento próximo à superfície.

Entre as técnicas mais comuns de digitalização estão incluídas a conversão AD direta, a digitalização por grade de contato e a digitalização por grade de proximidade.

A conversão AD direta é usada apenas pelos digitalizadores do tipo pantográfico ou de régua. O método mais simples é o do digitalizador de régua. Este tem dois braços articulados: um que se move na direção horizontal, e outro, na vertical. Potenciômetros são conectados a cada braço, por meio de engrenagens ou de polias. À medida que deslocamos os braços pela superfície, os potenciômetros giram, produzindo uma voltagem contínua proporcional à distância percorrida naquela direção, desde a origem. Desse modo, obtemos as coordenadas X e Y dos pontos.

O digitalizador pantográfico, por sua vez, possui dois braços, articulados em um "ombro" e um "cotovelo", cada um dispondo de um potenciômetro. As coordenadas X e Y, nesse caso, têm que ser calculadas por métodos trigonométricos, a partir dos ângulos de posicionamento de cada braço.

Um conversor AD, ou analógico-digital, é responsável pela conversão da voltagem gerada nos potenciômetros. O conversor AD utiliza um processo de *comparação escalonada*: uma voltagem de referência, gerada internamente no conversor AD, é aumentada passo a passo, em pequenos incrementos fixos. A cada passo, ela vai sendo comparada com o sinal de entrada. Quando, finalmente, as duas voltagens se igualam, o número de passos que foram necessários é enviado ao computador. A voltagem de referência é então zerada, e o processo recomeça para um novo ponto do sinal de entrada. A *freqüência de amostragem* corresponde ao número de vezes que isso é feito por segundo.

O digitalizador pantográfico requer dois conversores AD, um para cada potenciômetro. A resolução do conversor (ou precisão) é dada pelo número máximo de "partes" (valor do incremento) em que uma voltagem de entrada pode ser dividida, e está relacionada ao número de bits do conversor. Assim, um conversor de oito bits, por exemplo, pode gerar apenas um número entre 0 e 255 — ou seja, a voltagem máxima é dividida em 256 pedacinhos.

A informação gerada por cada conversor é transferida diretamente para a memória do computador, através de alguma porta de entrada (serial ou paralela). Se a freqüência de amostragem for muito grande, será necessário utilizar uma quantidade muito grande de memória. Por isso, a maioria dos programas gráficos não armazena toda a informação de um desenho em detalhes, mas simplesmente vai convertendo as coordenadas em linhas na tela.

## SENSORES DE SUPERFÍCIE

O funcionamento dos tabletes existentes para micros baseia-se, geralmente, em dois tipos distintos de mecanismo. Ambos dependem de sensores ocultos sob a superfície do tablete.

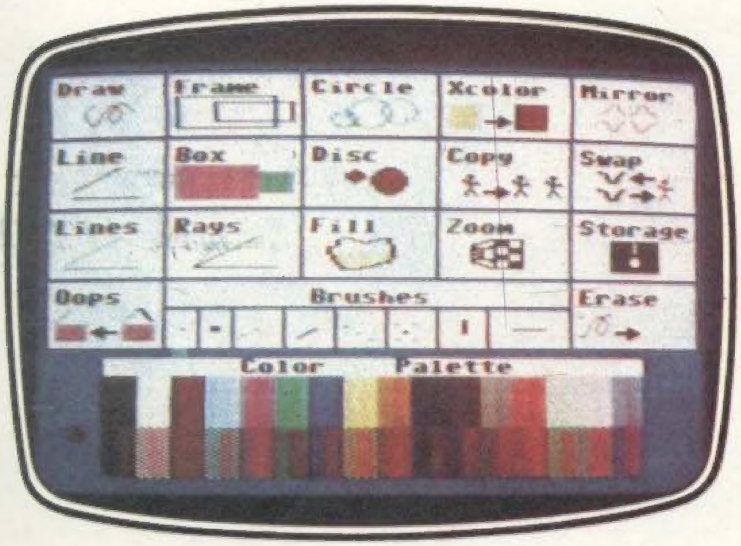
Os tipos sensíveis ao contato direto são formados por duas folhas de plástico. Uma delas contém fios paralelos, espaçados na direção horizontal; a outra, fios espaçados na direção vertical. Ao se encostar a ponta do estilete ou da caneta na superfície do tablete, o contato elétrico estabelecido entre os fios horizontais e verticais, naquele ponto, informa ao computador as coordenadas do mesmo. Pode-se reconhecer facilmente esse tipo de tablete, já que qualquer instrumento pontiagudo, ou mesmo o próprio dedo, é capaz de ativar o sistema.

As mesas digitalizadoras mais precisas, entretanto, utilizam o princípio da proximidade por indução eletromagnética. A malha de fios embutida sob a superfície recebe correntes elétricas em um sistema de varredura constante. Na ponta da caneta ou do cursor especial existe uma bobina detetora, que não precisa estar em contato direto com a superfície. Pode-se, assim, colocar sobre esta uma folha de papel com o desenho já feito e copiá-lo.

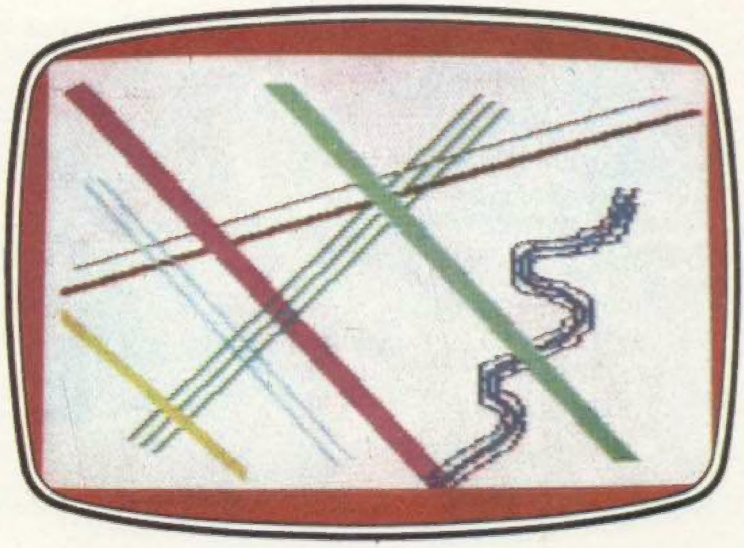
## O SOFTWARE

Embora sejam muitas as diferenças dos tabletes digitalizadores quanto ao hardware, o que realmente conta, em termos da exploração dos recursos de cada um, é o software. Como este também é o responsável pela definição das características do desenho e pela facilidade com que ele pode ser executado, a maior preocupação do usuário deve ser a escolha do programa aplicativo. Para que não se cometam erros, convém sempre assistir a uma demonstração completa do sistema.

Evidentemente, o que o software é capaz de fazer depende muito do modelo do seu computador. Quanto mais rápido ele for, e quanto maior a sua memória, mais espaço existirá para a operação de um tablete sofisticado. Aplicações em alta resolução, assim como programas com recursos embutidos, exigem muita memória. É interessante, portan-




No menu de abertura, o KoalaPad mostra as opções disponíveis.



A largura das "pinceladas" pode ser escolhida à vontade.

to, utilizar disquetes, que facilitam o armazenamento de telas e expandem a capacidade do sistema, pelo uso do disco como memória virtual.

O mesmo acontece com a capacidade gráfica do sistema. Um bom programa para tabletes gráficos explora até o limite máximo os recursos disponíveis no microcomputador.

Muitos programas controlados apenas pelo teclado, como o publicado no artigo da página 414, são similares, em operação, ao software destinado aos tabletes. Porém, quanto à facilidade, rapidez, conveniência e flexibilidade, é enorme a diferença entre um sistema operado por teclado e o operado por um tablete. Um sistema controlado por teclado não permite, por exemplo, o desenho a mão livre.

## COMO UTILIZAR UM TABLETE GRÁFICO

As combinações específicas entre computador/tablete variam bastante, pelas razões expostas anteriormente. Entretanto, todos os programas existentes foram projetados para trabalhar simulando o ato de desenhar ou pintar sobre uma folha de papel. É isto, mais do que qualquer outra coisa, que dá aos tabletes gráficos tanta vantagem em relação aos demais métodos de criação de imagens de alta resolução no microcomputador.

Em geral, um menu de abertura, como o existente para o KoalaPad, oferece ao usuário diversas opções quanto à técnica e tipo de desenho que se pretende executar. A escolha pode ser feita pelo teclado, ou, mais comumente, usando-se o próprio tablete gráfico como se fosse a paleta de um artista. A tela exibe potinhos de tinta, pincéis e outros materiais, que são selecionados com o cursor do tablete. Quando o cursor correspondente na tela estiver no lugar desejado, pressiona-se um botão ou tecla.

O desenho é executado na tela a mão livre, seja usando apenas a imaginação, seja seguindo um modelo feito sobre papel, que pode ser afixado ao tablete. Cabe ao programa receber os sinais gerados pelo tablete, interpretá-los e colocar instantaneamente a imagem traçada sobre a tela.

## OS PROGRAMAS E SEUS RECURSOS

Algumas vezes, é difícil traçar uma reta, ou uma figura geométrica regular, a mão livre. Para isso, a maioria dos programas possui um menu de curvas e de figuras geométricas — como triângulos, retângulos, arcos, círculos, elipses, retas etc. — que podem ser escolhidas, posicionadas sobre a tela e traçadas automaticamente, no tamanho que se desejar.

Os programas também oferecem o recurso de apagar e corrigir partes do desenho, ou, nos sistemas mais sofisticados, de voltar atrás, desfazendo-se a última operação realizada. Se isso não for suficiente para a obtenção do resultado pretendido, pode-se ainda recorrer ao recurso de apagar toda a tela e começar tudo de novo.

É possível mudar a cor usada nos traços do desenho, assim como no preenchimento automático de áreas delimitadas (*paint*), a qualquer instante. Deve-se, inicialmente, delinear o desenho utilizando uma determinada cor. Em seguida, coloca-se o cursor gráfico no interior da área assim demarcada, e a seleção adequada é realizada. Se o espaço não estiver completamente fechado, a cor "escapa" para fora do mesmo, podendo tomar toda a tela. Por isso, convém sempre completar o desenho antes de começar a colori-lo.

Opções mais especializadas variam de programa para programa. Alguns possibilitam o traçado automático de imagens especulares (cópia invertida), ou a cópia de uma parte do desenho em outra posição. Esta operação é chamada de "carimbo", pois permite ao usuário criar, por exemplo, uma floresta inteira a partir do desenho de uma única árvore, ou, para maior variação, de sua cópia especular.

## COMO GUARDAR UMA IMAGEM

Se você completou um desenho bonito e elaborado, certamente desejará armazená-lo permanentemente. Muitos softwares gráficos oferecem essa possibilidade por meio de um comando — dentro, é claro, das limitações impostas pelo hardware que você tem à disposição. As figuras armazenadas podem ser recuperadas depois, com um comando de leitura, e os recursos do programa permitirão, ainda, que você as modifique. Se quiser, combine-as com outras figuras, que podem ser obtidas da biblioteca de ícones, fornecidos junto com o sistema ou criados por você.

Armazene o desenho por tempo indeterminado ou utilize-o em outro programa — num jogo, por exemplo.

Para obter uma cópia da sua "obra de arte", fotografe a tela usando um filme em cores, ou copie o desenho em um *plotter* ou impressora gráfica.

A opção de "zoom" possibilita o desenho de pixels individuais.

Um recurso útil às composições é a repetição de elementos-chave.

# AVALANCHE: A ROTINA PRINCIPAL


- TAREFAS DA ROTINA PRINCIPAL
- ORIGEM
- ACERTOS INICIAIS
- DANDO A PARTIDA


**Já criamos uma página-título e uma página de instruções. Temos um tema musical e um cenário. Para completar o jogo, falta apenas uma rotina que controle as demais e dê vida a Willie.**

Este é o sétimo artigo da série *Avalanche* e você deve estar pensando, com razão, que já é hora de colocar o programa em funcionamento.

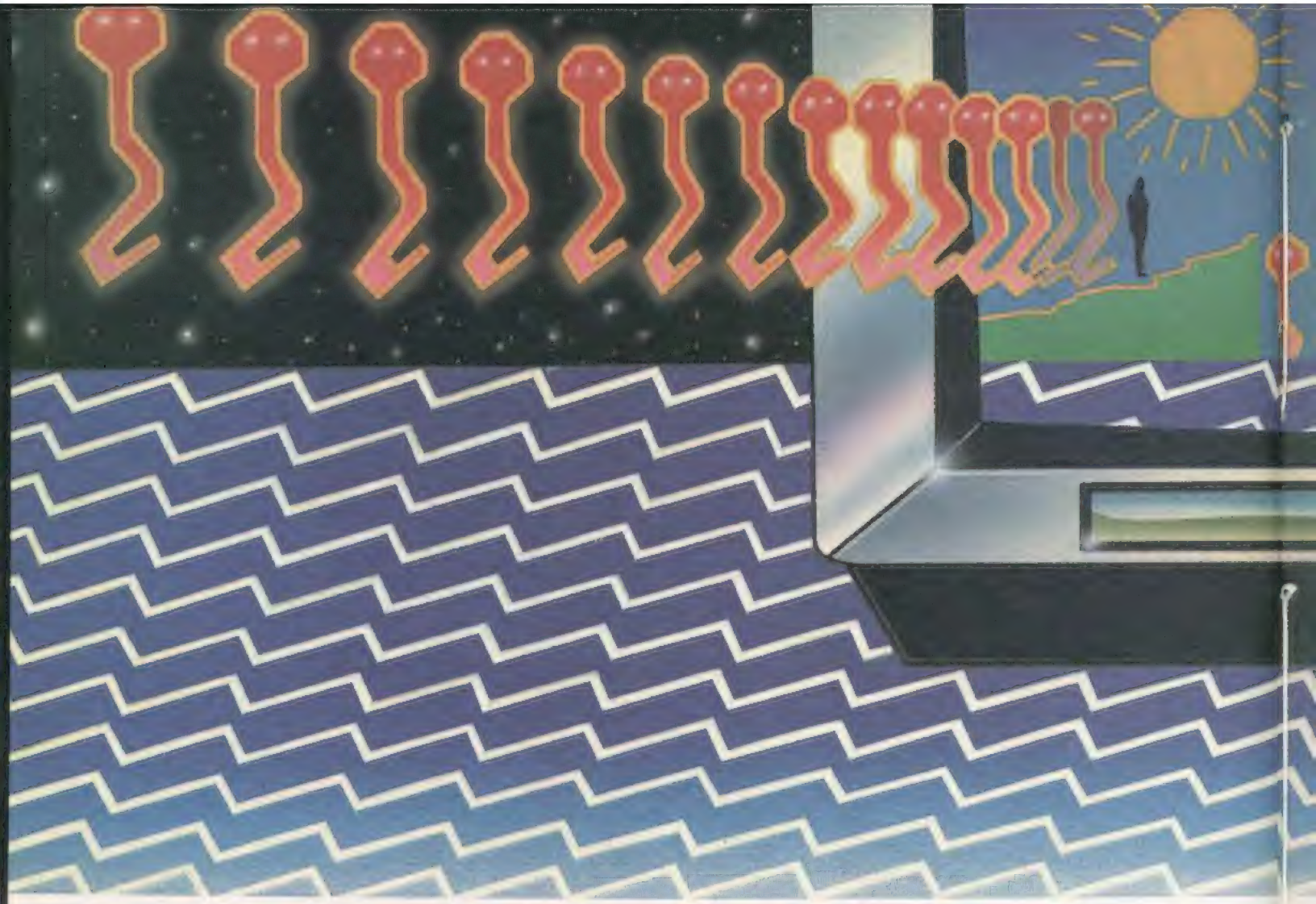
Um jogo em linguagem de máquina, porém, precisa de uma rotina principal que chame as rotinas de instrução, de execução musical e de criação do cenário. Essa rotina deve ainda acertar os valores das variáveis do jogo, tais como o placar, o número de vidas e o nível de dificuldade. Sua origem será o endereço a ser chamado quando quisermos rodar o programa.

Depois de pronto, o programa será executado automaticamente assim que for lido da fita cassete. Por enquanto, contudo, não há razão para isso. Além do mais, precisamos ter acesso aos códigos do programa para terminá-lo, o que seria absolutamente impossível se sua execução fosse automática.

## TAREFAS DA ROTINA PRINCIPAL

A listagem apresentada a seguir corresponde à rotina principal que dá início à versão do nosso jogo para os microcomputadores Spectrum.

```
 10 REM org 58576
 20 REM gbin call ti
 30 REM ld a,5
 40 REM ld (57343),a
 50 REM ld a,0
 60 REM ld (57344),a
 70 REM ld hl,0
 80 REM ld (57337),hl
 90 REM ld (57339),hl
100 REM ld (57341),hl
110 REM nlv ld a,19
120 REM ld (msk+1),a
130 REM ret
140 REM org 58035
150 REM ti *
160 REM org 58281
170 REM msk *
```



A origem é o endereço que chamaremos por intermédio do comando **RANDOM USR** para que o programa seja executado. Não se esqueça de anotá-lo.

Os rótulos que fazem parte da listagem não são chamados por esta seção do jogo. Eles serão utilizados por rotinas futuras, quando nosso personagem tiver morrido e você quiser começar o jogo novamente.

## ENDEREÇOS

O programa começa chamando a sub-rotina **ti**, que imprime os títulos e a página de instrução.

Vários parâmetros relativos à contagem de pontos são então estabelecidos. O escore propriamente dito é colocado nos endereços 57337 a 57342. O número de vidas fica em 57343 e o nível de dificuldade em 57344.

## ACERTOS INICIAIS

Para que o jogador tenha direito a cinco vidas, o número 5 é colocado em 57343, com o auxílio do acumulador. Zero é colocado do mesmo modo em 57344, fazendo com que o nível de dificuldade inicial seja 0.

O escore inicial, zero, é acertado colocando-se 0 nos endereços 57337, 57338, 57339, 57340, 57341 e 57342, por meio do par de registros HL.

Em seguida, A recebe o valor 19, e a rotina musical **msk** é ajustada para executar as dezenove primeiras notas da melodia *Greensleeves*.

## T

Esta é a rotina que dá início à versão do nosso jogo para o TRS-Color.

```
10   ORG 19426
20   GBIN JSR START
30   LDA #5
40   STA 18239
50   CLR 18238
60   LDX #18240
70   LDB #6
80   GBINI CLR ,X+
90   DECB
100   BNE GBINI
110   RTS
120   START EQU 19000
```

A origem corresponde ao endereço a ser chamado por intermédio da instrução **EXEC**, quando quisermos iniciar a execução do programa. Não se esqueça de anotar esse número.

Os rótulos da listagem não são utilizados nesta parte da rotina. Eles serão chamados por outras seções do programa, que publicaremos mais tarde. Sua função é recomeçar o jogo, caso você queira prosseguir quando Willie for fatalmente vitimado.

## DANDO A PARTIDA

Antes de mais nada, o programa chama a sub-rotina **START**, responsável pela impressão da página-título e da página de instruções, que abrem o jogo. Em seguida, são estabelecidos diversos valores iniciais que influem na contagem final de pontos.

O nível de dificuldade do jogo é armazenado no endereço de memória 18238. O número de vidas, por sua vez, fica em 18239. O escore propriamente dito é guardado em seis bytes, a partir do endereço 18240.

## PRIMEIROS ACERTOS

O número inicial de vidas de Willie, 5, é colocado no acumulador e transferido para 18239. A instrução **CLR** limpa a posição de memória 18238, para zerar o nível de dificuldade.

O registro X recebe o endereço inicial dos bytes de escore e o contador B recebe o número 6 — número de bytes de escore. A instrução **CLR ,X+** limpa o conteúdo da posição de memória apontada por X e soma uma unidade ao valor de X. **DEC B** diminui B em uma unidade. A instrução **BNE** retorna ao rótulo **GBINI** para apagar o próximo byte até que B seja zero. Esse pequeno laço limpa todas as posições de 18240 a 18245 — o que equivale a zerar todos os endereços que se referem ao escore, um depois do outro.

Esta é a rotina principal que dá início ao jogo na versão do MSX.

```
10    org -11670
20    oin call -12288
30    call -12166
40    ld a,5
50    ld (-5221),a
60    ld a,0
70    ld (-5228),a
80    ld hl,0
90    ld (-5219),hl
100    ld (-5217),hl
110    ld (-5215),hl
120    hlv ld a,19
130    ld (-12162),a
140    ret
150    end
```

A origem dessa rotina é o endereço que você chamará por intermédio de dois comandos, **DEFUSR** e **USR(0)**, para começar a execução do jogo. Não se esqueça de anotar esse número.

Os rótulos que precedem as instruções não são chamados nesta parte do programa. Serão utilizados pelas rotinas que inicializam o jogo quando o pobre Willie tiver sido morto e você quiser jogar de novo.

## DANDO A PARTIDA

Inicialmente a rotina chama as sub-rotinas -12228 e -12166. Enquanto a primeira imprime o título e a página de instruções, à segunda cabe executar a música *Greensleeves*.

Os vários parâmetros da contagem de pontos são, então, inicializados. O escore propriamente dito é colocado nos endereços -5219 até -5214. O número de vidas fica em -5221 e o nível de dificuldade, em -5228.

## ACERTOS INICIAIS

O número 5 é colocado em -5221, por meio do acumulador, para que o jogador inicie a partida dispondo de cinco vidas. Do mesmo modo, 0 é colocado em -5228, para que o nível de dificuldade inicial seja igual a 1.

Os endereços -5219, -5218, -5217, -5216, -5215 e -5214, por sua vez, são preenchidos com o valor 0, por intermédio do par de registros HL, zerando o escore inicial.

Em seguida, a rotina responsável pela música — colocada no endereço -12166 — é ajustada para executar exclusivamente as dezenove primeiras notas da melodia *Greensleeves*.

# ROTINA EM CÓDIGO DE MÁQUINA (1)

**Programas sofisticados e rápidos exigem a combinação de código de máquina e BASIC. Aprenda aqui uma série de truques para tornar mais eficaz a interação entre ambos.**

As mais modernas versões do interpretador BASIC desenvolvido pela empresa norte-americana Microsoft foram implementadas nos microcomputadores das linhas TRS-80, TRS-Color, MSX e IBM-PC. Tanto elas quanto o MBASIC (usado em qualquer micro que tenha o sistema operacional compatível com o CP/M) possuem diversos recursos para dar acesso direto às posições absolutas de memória do computador, bem como à execução de programas em código de máquina a partir de um programa em BASIC.

Já estudamos os principais elementos do BASIC para efetuar essas funções: o **PEEK** e o **POKE**. Porém, para uma interação mais eficaz entre programas em linguagem de máquina e programas em BASIC, outros comandos são necessários. Destacam-se, entre eles, o **DEFUSR**, o **USR** e o **VARPTR**.

Em artigos anteriores, vimos diversos exemplos de como executar um programa em linguagem de máquina a partir de um programa em BASIC. O método mais comum consiste em armazenar o programa em código de máquina em uma parte não usada da memória RAM (ou seja, em uma parte não sujeita a ser invadida pelo programa BASIC ou por sua área de variáveis), e executá-lo por meio de um comando **USR** colocado em um ponto qualquer do programa em BASIC. O comando **DEFUSR**, ou equivalente, é usado para indicar a locação inicial do programa em linguagem de máquina.

Esse método, entretanto, apresenta alguns problemas:

- O programa em código de máquina precisa ser montado à parte e armazenado em fita ou disco, para, então, ser carregado na memória.
- É necessário reservar uma parte protegida da memória para armazenar o programa, o que pode ser inconveniente em muitos tipos de aplicação.

## A FUNÇÃO VARPTR

Examinaremos, neste artigo, uma técnica alternativa, que consiste na utilização da função **VARPTR** para armazenar programas em linguagem de máquina dentro do próprio programa em BASIC. Essa técnica proporciona poderosos recursos de programação para os usuários das linhas TRS-80, TRS-Color e MSX (os micros das linhas Sinclair e Apple dispõem do comando **CALL**, equivalente ao **USR**, mas não têm a função **VARPTR**). Veremos aqui como empregar a função **VARPTR** em micros da linha TRS-80; em artigos futuros trataremos do TRS-Color e do MSX.

O "truque" é muito simples: em vez de armazenarmos um programa em código de máquina numa parte reservada de memória, nós o "enxertamos" dentro do programa em BASIC que o utilizará. Assim, quando o programa em BASIC for carregado da fita ou disco, para ser executado, trará junto, automaticamente, o programa ou programas em linguagem de máquina. Esse método permite a definição e armazenamento do número que quisermos de rotinas em linguagem de máquina, cada qual bem identificada dentro do programa em BASIC.

Mas onde armazenar código de máquina dentro de um programa em BASIC? Existem diversas técnicas disponíveis. As mais flexíveis, entretanto, usam o seguinte recurso: uma constante alfanumérica (por exemplo, **PG$**) é definida dentro do programa, com um comprimento correspondente ao número de bytes do programa em código de máquina. Suponhamos, por exemplo, que a rotina em código de máquina terá doze bytes de extensão. Definimos, então:

```
20 PG$="            "
```

Observe que foram colocados doze espaços em branco entre as aspas da linha 20. No lugar desses espaços, porém, poderiam introduzir quaisquer caracteres, iguais ou diferentes. O que vale é o número correto de caracteres. Muitos programadores preferem usar uma seqüência numérica para assegurar que o número de bytes seja correto:

```
20 PG$="123456789012"
```

Feito isso, carregamos os bytes disponíveis (que estão automaticamente fixos e reservados na área de memória de programa) na variável **PG$**, com os códigos decimais, binários, octais, ou hexadecimais correspondentes ao programa em código de máquina.

Um pequeno programa, contendo um laço **FOR...NEXT**, pode se encarregar dessa tarefa. Ele lerá os códigos, armazenados em linhas **DATA**, e os colocará nas locações absolutas de memória, correspondentes à variável **PG$**, usando comandos **POKE**.

A operação só precisa ser feita uma vez. Depois, o programa de carregamento pode ser apagado.

O processo é simples, mas há um detalhe técnico a ser resolvido: a localização das posições da memória onde **PG$** está armazenada. Teoricamente, podemos fazer esse cálculo para qualquer programa, desde que saibamos a locação absoluta da memória onde ele começa (geralmente fixa para cada tipo de computador), e quantos bytes gasta cada linha do programa até chegarmos à linha em que **PG$** é definida.

Na prática, porém, tal cálculo é difícil e demorado, podendo tornar-se realmente trabalhoso se o programa ainda estiver em desenvolvimento. Nesse caso, cada alteração no programa, feita acima da linha onde a variável **PG$** é definida, altera a sua locação na memória.

## ONDE ESTÃO AS VARIÁVEIS?

Felizmente, é possível encarregar o próprio programa de calcular automaticamente a localização de **PG$** na memória. Para isso, devemos usar a função **VARPTR**, cujo nome deriva da expressão inglesa *VARiable PoinTeR*, ou apontador de variável.

Para entender como essa função trabalha, precisamos saber como o computador armazena cadeias alfanuméricas na memória. Como vimos no artigo da página 947, seu programa pode ter acesso direto a toda essa informação por meio da função VARPTR.

Recapitulando: toda vez que uma variável alfanumérica é definida, o interpretador BASIC cria três bytes *apontadores*, que contêm o comprimento da cadeia alfanumérica e o endereço absoluto de onde ela se inicia. Por exemplo, para uma variável **A$**:

- O USO DE ROTINAS EM CÓDIGO DE MÁQUINA DENTRO DE PROGRAMAS BASIC
- OS COMANDOS **USR** E **DEFUSR**
- COMO ARMAZENAR ROTINAS
- **USR** EM UM PROGRAMA
- DIFERENÇAS ENTRE O BASIC PARA CASSETE E PARA DISCO
- UM PROGRAMA PARA PREENCHER A TELA

**VARPTR(A$)** = locação do apontador de **A$**
**PEEK(VARPTR(A$))** = comprimento de **A$**
**PEEK(VARPTR(A$)+1)** = byte menos significativo do endereço de **A$**
**PEEK(VARPTR(A$)+2)** = byte mais significativo do endereço de **A$**

## UM PROGRAMA COMPLETO

Sabendo como determinar o endereço inicial de uma variável *string* na memória, fica fácil escrever um programa para carregar código de máquina na mesma. Vejamos, por meio de um exemplo, como isso funciona.

A rotina em linguagem de máquina, apresentada a seguir, preenche toda a tela com um caractere qualquer, previamente colocado na primeira posição da tela (por exemplo, se colocarmos * na posição 0 da tela, uma chamada à rotina preencherá a tela com asteriscos quase que instantaneamente).

```
10 ORG 0BFF0H     ;ORIGEM
20 LD HL,15360    ;HL APONTA P/0
30 LD DE,15361    ;DE APONTA P/1
40 LD BC,1023     ;POR 1023 VEZES
50 LDIR           ;REPETE
60 RET            ;VOLTA AO BASIC
70 END
```

Quando traduzido por um Assembler, esse curto programa daria doze bytes em hexadecimal:

```
Linha 20:  21  00  3C
Linha 30:  11  01  3C
Linha 40:  01  FF  03
Linha50:   ED  B0
Linha 60:  C9
```

que, em decimal, correspondem a:

```
33,0,60,17,1,60,1,255,3,237,176
,201
```

Vamos escrever agora nosso programa principal — ou seja, o programa em linguagem BASIC que utiliza a rotina em código de máquina.

Não rode o programa ainda, pois o código de máquina não foi carregado:

```
10 CLS:PRINT"PROGRAMA DE TESTE
20 PG$="                "
30 D=PEEK(VARPTR(PG$)+1) + 256
* PEEK(VARPTR(PG$)+2)
40 DEFUSR0=D
80 PRINT:INPUT "CARACTERE ";C$
90 CLS:PRINT C$
100 X=USR0(0)
110 FOR I=1 TO 1000:NEXT I
120 GOTO 10
```

O programa funciona da seguinte maneira: a linha 20 define um *string* fixo, **PG$**, na memória de programa, com o comprimento de doze bytes.

As linhas 30 e 40 localizam o endereço da variável **PG$** e o definem como início de uma rotina de máquina.

O comando usado para isso denomina-se **DEFUSR**, que é uma abreviatura da expressão *DEFine USeR function*. Seu objetivo é informar ao programa principal o endereço absoluto inicial da rotina. Nos microcomputadores da linha TRS-80 com BASIC para disco, podem-se definir até dez rotinas de usuário simultaneamente, por meio dos comandos **DEFUSR0** a **DEFUSR9**, que corresponderão às chamadas **USR0** a **USR9**.

Para os microcomputadores da linha TRS-80 com BASIC para cassete, pode-se definir apenas uma rotina **USR**. Nesse caso, não se pode contar com o comando **DEFUSR** para informar o seu endereço de partida. Temos, assim, que dar dois **POKE** em um apontador específico da memória de trabalho do computador. Eles correspondem aos bytes de endereço 16526 (para o byte menos significativo) e 16527 (para o byte mais significativo). Esse processo torna possível copiar diretamente os bytes obtidos pela função **VARPTR**. Para computadores com BASIC para cassete, portanto, devemos substituir as seguintes linhas no programa anterior:

```
30 POKE 16526,PEEK(VARPTR(
PG$)+1)
40 POKE 15527,PEEK(VARPTR(
PG$)+2)
```

Como a variável **PG$** está fixa dentro do programa, a definição com **DEFUSR** ou com **POKE** só precisa ser realizada uma vez.

A linha 80 pede ao usuário que entre pelo teclado o caractere com que deseja preencher a tela. A linha 90 limpa a mesma e coloca o caractere indicado na primeira posição.

Finalmente, a linha 100 executa a rotina, por meio da função fictícia **USR** (da expressão, em inglês, *USeR function*). Como não se trata de um comando, **USR** pode ser colocada dentro de uma expressão matemática válida. No exemplo, não precisamos passar nenhum argumento para a rotina de máquina (o que seria feito por meio do valor entre parênteses), nem obter um valor de retorno (o que seria feito através do próprio valor da função **USR**). Como vimos anteriormente, nos microcomputadores com BASIC para disco, cada **DEFUSRn** (onde **n** é um número inteiro entre 0 e 9) corresponde à rotina **USRn**.

Como os computadores com linguagem BASIC para cassete aceitam apenas uma rotina de usuário, devemos substituir a linha 100 por:

```
100 X=USR(0)
```

## PROGRAMA DE CARREGAMENTO

Antes de usar ou gravar o programa principal, é necessário carregar o programa de código de máquina na variável **PG$**. Para fazê-lo, anexe ao programa anterior as seguintes linhas:

```
50 FOR I%=D TO D+12:READ N% :
POKE I%,N% : NEXT I%
60 DATA 33, 0, 60, 17, 1, 60,
1, 255, 3, 237, 176, 201
70 STOP
```

O programa pode ser apagado com:

```
DELETE 50-70
```

Você notará que, ao listar o programa principal, a linha 30 não estará mais vazia: ela conterá códigos gráficos e até mesmo comandos em linguagem BASIC, dependendo da linha do computador que você estiver usando.

Você pode agora gravar e executar o programa quantas vezes quiser, sem ter que carregar separadamente o programa em linguagem de máquina. E, é claro, até dez variáveis alfanuméricas diferentes poderão ser usadas no mesmo programa. Elas definirão rotinas de máquina separadas, que serão chamadas de **USR0**, **USR1**, **USR2** etc.

# A ARANHA MARCIANA (2)

**O jogo está completo. Armado com suas flechas, Freddy observa os balões que começam a subir. Precisará de muito sangue frio para estourá-los e impedir que a aranha venha em sua direção.**

Já digitamos as rotinas encarregadas de inicializar o jogo e de fazer a montagem dos gráficos.

Vamos agora completar o programa, adicionando as rotinas de animação.

## O LAÇO PRINCIPAL

```
 10 CLEAR 65287
 20 GOSUB 1000
 30 GOSUB 3000
 50 IF ax<>29 THEN  GOSUB 300
 70 GOSUB 400
 90 GOSUB 500
100 GOSUB 200: IF dead=0 THEN
GOTO 50
```

```
10 CLS 3: PRINT @266,"INICIALIZ
ANDO";:SCREEN 0,1
20 GOSUB 1000
25 GOSUB 1600
30 GOSUB 3000
50 IF AX<>29 THEN GOSUB 300
70 GOSUB 400
90 GOSUB 500
100 GOSUB 210:IF DD=0 THEN 50
```

```
10 CLS:COLOR 1,14,14
15 OPEN "GRP:" FOR OUTPUT AS #1
20 GOSUB 1000
30 GOSUB 3000
50 IF AX<>29 THEN GOSUB 300
70 GOSUB 400
90 GOSUB 500
100 GOSUB 210:IF DD=0 THEN 50
```

```
10  TEXT : HOME : PRINT "UM MOM
ENTO..."
15  HIMEM: 18817
20  GOSUB 1000
30  GOSUB 3000
50  IF AX <  > 28 THEN  GOSUB 3
00
70  GOSUB 400
90  GOSUB 500
100  GOSUB 210: IF DD = 0 THEN
50
```

Esse programa, o principal do jogo, foi estruturado de modo a chamar apenas as sub-rotinas necessárias. Primeiramente, ele define a tela e reserva espaço na parte superior da memória (Apple, TK-2000 e Spectrum). As sub-rotinas chamadas nas linhas 20 a 30 (listadas no artigo anterior) têm a função de definir os blocos gráficos e inicializar o valor do recorde.

O laço principal do programa vai da linha 50 à linha 100, e se repete enquanto Freddy estiver vivo (variável **dead** ou **DD** maior do que zero). Esse laço é responsável pelo movimento da flecha, da aranha, de Freddy e dos balões. Cada rotina chamada atualiza as variáveis dentro do laço.

## HORA DO "ALMOÇO"

```
105 LET s(xinc)=1
110 FOR x=s(xpos) TO 29: GOSUB
500: NEXT x
120 LET s(yinc)=1: LET s(xinc)
```

- O JOGO COMPLETO
- LAÇO PRINCIPAL
- A ARANHA ATACA FREDDY
- BALÕES EM MOVIMENTO
- FLECHAS CONTRA BALÕES
- FREDDY SOBE E DESCE A ESCADA
- A ANIMAÇÃO DA ARANHA
- ESTOURANDO BALÕES

```
=0
125 FOR y=s(ypos) TO 19
130 IF y=my AND ax=29 THEN
POKE 23607,60: PRINT AT my+1,
29;" ": POKE 23697,252
140 GOSUB 500
150 NEXT y
160 POKE 23607,60: PRINT AT 10
,0; INK 2; PAPER 7; BRIGHT 1;
FLASH 1;"Voce esta morto!  Out
ra jogada ?";: POKE 23607,252
165 LET a$=INKEY$: IF a$=""
THEN  GOTO 160
170 IF a$="s" OR a$="S" THEN
GOTO 30
175 IF a$<>"n" AND a$<>"N"
THEN  GOTO 160
180 POKE 23607,60: CLS : STOP
```

```
105 S(XI)=1
110 FOR X=S(XP) TO 29:GOSUB 500
:NEXT X
120 S(YI)=1:S(XI)=0
125 FOR Y=S(YP) TO 19
130 IF Y=MY AND AX=29 THEN PUT(
232,(MY+1)*8)-(239,(MY+1)*8+7),
S1,PSET
140 GOSUB 500
150 NEXT Y
160 FOR SL=180 TO 160 STEP -1:S
OUND SL,1:NEXT:CLS:PRINT @256,"
 VOCE ESTA MORTO !
  QUER RECOMECAR (S/N) ?"
165 A$=INKEY$:IF A$="" THEN 165
170 IF A$="S" THEN 30
175 IF A$<>"N" THEN 160
180 CLS:END
```

```
105 S(XI)=1
110 FOR X=S(XP) TO 29:GOSUB 500
:NEXT X
120 S(YI)=1:S(XI)=0
125 FOR Y=S(YP) TO 19
130 IF Y=MY THEN PUT SPRITE 2,(
X2,Y2),0,FD:PUT SPRITE 3,(X2,Y2
),0,FL
140 GOSUB 500
150 NEXT Y
160 PLAY"ABCDEF":FOR X=1 TO 300
:NEXT:SCREEN0:CLS:PRINT"Você es
tá morto!":LOCATE 0,12:PRINT"Jo
ga novamente? (S/N)"
165 A$=INKEY$:IF A$="" THEN 165
170 IF A$="S" THEN RUN
175 IF A$<>"N" THEN 160
180 CLS:END
```

```
105 S(XI) = 1
110  FOR X = S(XP) TO 29: GOSUB
 500: NEXT X
120 S(YI) = 1:S(XI) = 0
125  FOR Y = S(YP) TO 19
130  HCOLOR= 0: FOR X = 16 TO 2
4: HPLOT 230,Y * 8 + X TO 260,Y
 * 8 + X: NEXT : HCOLOR= 3
140  GOSUB 500
150  NEXT Y
160  FOR SL = 1 TO 5: PRINT  CH
R$ (7);: NEXT : TEXT : HOME : P
RINT "VOCE ESTA MORTO!": VTAB 1
5: PRINT "QUER RECOMECAR? (S/N)
";
170  GET A$
180  IF A$ = "S" THEN  HOME : G
OTO 30
190  HOME : END
```

Após a remoção de todas as portas da gaiola da aranha, a variável **dead** (ou **DD**, nos outros microcomputadores) assume o valor 1 e as linhas 105 a 180 são executadas (190 no Apple e no TK-2000).

A aranha move-se horizontalmente até chegar logo acima de Freddy. Em seguida, realiza um movimento vertical, alcançando nosso infeliz personagem, que é, então, devorado. Quando isso ocorre, o jogador tem a opção de jogar mais uma vez (linhas 160 a 180). No programa do Spectrum, deve-se colocar o valor 60 na posição 23607 para que todo o conjunto de caracteres seja usado novamente.

## SOBEM OS BALÕES

```
210 LET b(count)=b(count)-1:
IF b(count)<>0 THEN  GOTO 280
220 LET b(count)=b(maxcount):
PRINT AT b(ypos)+1,b(xpos);"
";: LET b(ypos)=b(ypos)-1: IF
b(ypos)=4 THEN  GOSUB 600:
POKE 23607,60: PRINT AT 1,10+(
3-props)*9;"";AT 2,10+(3-props
)*9;" ": POKE 23607,252: LET
props=props-1
225 IF props=0 THEN  LET dead=
1
230 IF ((ay<>b(ypos) AND ay<>b
(ypos)+1) OR (ax<b(xpos)-1 OR
ax>b(xpos)+1)) THEN  GOTO 250
240 LET score=score+b(points):
GOSUB 600: IF score>hiscore
THEN  LET hiscore=score: POKE
23607,60: PRINT AT 0,23; INK 0
; PAPER 6;hiscore: POKE 23607,
252
245 GOTO 380
250 GOSUB 4300
280 RETURN
```

```
210 B(CT)=B(CT)-1:IF B(CT)<>0 T
HEN 280
220 B(CT)=B(MC)
221 B(YP)=B(YP)-1: IF B(YP)=4 T
HEN X2=B(XP)*8:Y2=B(YP)*8+8:PUT
(X2,Y2)-(X2+15,Y2+15),SP,PSET:G
OSUB 600:X2=(10+(3-PP)*9)*8:PUT
 (X2,8)-(X2+7,15),S1,PSET:PUT(X
2,16)-(X2+7,23),S1,PSET:PP=PP-1
225 IF PP=0 THEN DD=1
230 IF ((AY<>B(YP) AND AY<>B(YP
)+1) OR (AX<B(XP)-1 OR AX>B(XP)
+1)) THEN 250
240 SC=SC+B(PO):X2=B(XP)*8:Y2=(
B(YP)+1)*8:PUT(X2,Y2)-(X2+15,Y2
+15),SP,PSET:GOSUB 600:IF SC>HS
 THEN HS=SC:GOSUB 1700
245 GOTO 400
250 GOSUB 4300
280 RETURN
```

```
210 B(CT)=B(CT)-1:IF B(CT)<>0 T
HEN 280
220 B(CT)=B(MC)
222 B(YP)=B(YP)-1:IF B(YP)=4 TH
EN GOSUB 600:LINE ((10+((3-PP)*
9))*8,7)-((10+((3-PP)*9))*8+7,2
4),14,BF:PP=PP-1
225 IF PP=0 THEN DD=1
230 IF ((AY<>B(YP) AND AY<>B(YP
)+1) OR (AX<B(XP)-1 OR AX>B(XP)
+1)) THEN 250
240 SC=SC+B(PO):GOSUB 600:IF SC
>HS THEN HS=SC:GOSUB 1700
245 GOTO 400
250 GOSUB 4300
280 RETURN
```

```
210 B(CT) = B(CT) - 1: IF B(CT)
 <  > 0 THEN 280
220 B(CT) = B(MC)
222 B(YP) = B(YP) - 1: IF B(YP)
 = 4 THEN X2 = B(XP) * 8:Y2 = B
(YP) * 8 + 8: HCOLOR= 0: DRAW B
A AT X2,Y2: GOSUB 600: HCOLOR=
0: FOR X = 0 TO 6: HPLOT X + (4
 - PP) * 77,4 TO X + (4 - PP) *
 75,22: NEXT :PP = PP - 1
225  IF PP = 0 THEN DD = 1
230  IF ((AY <  > B(YP) AND AY
<  > B(YP) + 1) OR (AX < B(XP)
- 1 OR AX > B(XP) + 1)) THEN 25
0
240 SC = SC + B(PO):X2 = B(XP)
* 8:Y2 = (B(YP) + 1) * 8: HCOLO
R= 0: DRAW BA AT X2,Y2: HCOLOR=
 3: GOSUB 600: IF SC > HS THEN
HS = SC: GOSUB 1700
245  GOTO 400
250  GOSUB 4300
280  RETURN
```

Os elementos mais importantes da matriz do balão são **b (count)** e **b (maxcount)**, para o Spectrum, e **B (CT)** e **B (MC)**, para as outras linhas. Cada vez que a rotina é acionada, a linha 210 decrementa **b (count)** ou **B (CT)**, quando ela atinge 0, o balão se move. Em segui-

da, a linha 220 copia o número de **b (maxcount)** para **b (count)** — ou **B (MC)** para **B (CT)**, nos demais computadores. A velocidade do balão é determinada pelo valor de **b (maxcount)** — ou **B (MC)**. A linha 220 verifica se o balão foi atingido ou se chegou ao topo da tela.

Se o balão foi atingido, o placar é incrementado; se ele chegou ao topo, uma porta é removida. Quando todas as portas forem removidas, a variável **dead** (ou **DD**) assume o valor 1.

## O ESTOURO DO BALÃO

```
300 PRINT AT ay,ax;"  ": LET
ax=ax-1: IF ax<0 THEN  LET ax=
29: PRINT AT my+1,29;"e": LET
ay=my+1: RETURN
310 IF ((ay=b(ypos) OR ay=b(yp
os)+1) AND (ax=b(xpos) OR ax=b
(xpos)+1)) THEN  LET score=sco
re+b(points): GOSUB 600: IF sc
ore>hiscore THEN  LET hiscore=
score: POKE 23607,60: PRINT AT
 0,23; INK 0; PAPER 6;hiscore:
 POKE 23607,252
330 IF ax<>29 THEN  GOSUB 4100
340 RETURN
```

```
300 PUT(AX*8,AY*8)-(AX*8+15,AY*
8+7),S2,PSET:AX=AX-1:IF AX<0 TH
EN AX=29:PUT(232,(MY+1)*8)-(239
,(MY+1)*8+7),E,PSET:AY=MY+1:RET
URN
310 IF ((AY=B(YP) OR AY=B(YP)+1
) AND (AX=B(XP) OR AX=B(XP)+1))
 THEN SC=SC+B(PO):X2=B(XP)*8:Y2
=B(YP)*8+8:PUT(X2,Y2)-(X2+15,Y2
+15),SP,PSET:GOSUB 600:IF SC>HS
 THEN HS=SC:GOSUB 1700
330 IF AX<>29 THEN GOSUB 4110
340 RETURN
```

```
300 AX=AX-1:IF AX<0 THEN AX=29:
PUT SPRITE 3,(232,(MY)*8),4,FL:
AY=MY:RETURN
310 IF ((AY=B(YP) OR AY=B(YP)+1
) AND (AX=B(XP) OR AX=B(XP)+1))
 THEN SC=SC+B(PO):GOSUB 600:IF
SC>HS THEN HS=SC:GOSUB 1700
330 IF AX<>29 THEN GOSUB 4110
340 RETURN
```

```
300  HCOLOR= 0: DRAW FL AT AX *
 8,AY * 8: HCOLOR= 3:AX = AX -
1: IF AX < 0 THEN AX = 28: DRAW
 FL AT AX * 8,(MY + 1) * 8:AY =
 MY + 1: RETURN
310  IF ((AY = B(XP) OR AY = B(
YP) + 1) AND (AX = B(XP) OR AX
= B(XP) + 1)) THEN SC = SC + B(
PO):X2 = B(XP) * 8:Y2 = B(YP) *
 8: HCOLOR= 0: DRAW BA AT X2,Y2
: HCOLOR= 3: GOSUB 600: IF SC >
 HS THEN HS = SC: GOSUB 1700
330  IF AX <  > 28 THEN  GOSUB
4110
340  RETURN
```

Esta e a rotina que anima a flecha. Ela apaga a imagem anterior e coloca a nova na posição seguinte, determinada

pela variável **AX**. Essa variável é decrementada na linha 300. Para evitar que a flecha seja posta fora da tela, **AX** é recolocada em 29 (28, no Apple e no TK-2000) sempre que atinge um valor inferior a zero. Quando o valor de **AX** é 29 (ou 28), a flecha está com Freddy, e, como o balão não pode ser estourado, a rotina é abandonada.

Logo depois do disparo da flecha — **AX<>29** —, a linha 310 verifica se ela atingiu o balão e, em caso positivo, aumenta o placar. A rotina da linha 600, que promove o estouro do balão, é, então, chamada.

A linha 330, por sua vez, chama a sub-rotina que desenha a flecha na posição de Freddy.

## FREDDY NA ESCADA

```
400 LET a$=INKEY$: IF a$="" 
THEN  RETURN
410 IF a$="Z" OR a$="z" THEN 
GOTO 450
420 IF a$="c" OR a$="C" THEN 
GOTO 440
430 IF a$<>" " THEN  RETURN
432 IF ax<>29 THEN  RETURN
434 LET ax=28: PRINT AT ay,29;
" ": RETURN
440 IF my=19 THEN  RETURN
```

```
445 PRINT AT my,30; INK 6;"kl"
: LET my=my+1: PRINT AT ay,29;
" ": IF ax=29 THEN  LET ay=ay+
 1
446 GOTO 470
450 IF my=5 THEN  RETURN
460 PRINT AT my+2,30; INK 6;"
kl": LET my=my-1: PRINT AT ay,
29;" ": IF ax=29 THEN  LET ay=
ay-1
470 GOSUB 4000: RETURN
```

```
400 IF PEEK(337)=255 THEN RETUR
N
410 IF PEEK(341)=247 THEN 450
420 IF PEEK(342)=247 THEN 440
430 IF PEEK(345)<>247 THEN RETU
RN
432 IF AX<>29 THEN RETURN
434 AX=28:PUT(232,AY*8)-(239,AY
*8+7),S1,PSET:RETURN
440 IF MY=19 THEN RETURN
445 PUT(240,MY*8)-(255,MY*8+7),
KL,PSET:MY=MY+1:PUT(232,AY*8)-(
239,AY*8+7),S1,PSET:IF AX=29 TH
EN AY=AY+1
446 GOTO 470
450 IF MY=5 THEN RETURN
460 PUT(240,MY*8+16)-(255,MY*8+
23),KL,PSET:MY=MY-1:PUT(232,AY*
8)-(239,AY*8+7),S1,PSET:IF AX=2
9 THEN AY=AY-1
470 GOSUB 4000:RETURN
```

```
400 A$=INKEY$
410 IF A$=CHR$(30) THEN 450
420 IF A$=CHR$(31) THEN 440
430 IF A$<>CHR$(32) THEN RETURN
432 IF AX<>29 THEN RETURN
434 AX=28:RETURN
440 IF MY=21 THEN RETURN
445 MY=MY+1:IF AX=29 THEN AY=AY
+1
446 GOTO 470
450 IF MY=5 THEN RETURN
460 MY=MY-1:IF AX=29 THEN AY=AY
-1
470 GOSUB 4000:RETURN
```

```
400  POKE  - 16368,0: FOR X = 1
 TO 10: IF  PEEK ( - 16384) > 1
27 THEN X = 10: NEXT :PK =  PEE
K ( - 16384): GOTO 410
405  POKE  - 16368,0: NEXT : RE
TURN
410  IF PK = 218 THEN 440
420  IF PK = 193 THEN 450
430  IF PK <  > 160 THEN  RETUR
N
432  IF AX <  > 28 THEN  RETURN

434 AX = 27: HCOLOR= 0: DRAW FL
 AT 224,AY * 8: HCOLOR= 3: RETU
RN
440  IF MY = 19 THEN  RETURN
445  HCOLOR= 0: DRAW FD AT 240,
MY * 8:MY = MY + 1: IF AX = 28
THEN  DRAW FL AT AX * 8,AY * 8:
AY = AY + 1
446  HCOLOR= 3: DRAW FD AT 240,
MY * 8
448  GOTO 470
450  IF MY = 5 THEN  RETURN
460  HCOLOR= 0: DRAW FD AT 240,
MY * 8:MY = MY - 1: IF AX = 28
THEN  DRAW FL AT AX * 8,AY * 8:
AY = AY - 1
465  DRAW FD AT 240,MY * 8
470  HCOLOR= 3: GOSUB 4000: RET
URN
```

Em todos os programas, as linhas 400 a 420 lêem o teclado, enquanto as linhas 430 e 440 verificam se a barra de espaços foi pressionada e se Freddy está com a flecha.

Enquanto nosso personagem se movimenta para cima ou para baixo na escada, os caracteres desta têm que ser repostos (ou a escada acabará desaparecendo). Se **AX** = 29 (28, na versão para o Apple e o TK-2000), a flecha também deve ser movimentada.

Para que o programa funcione adequadamente no TK-2000, os usuários precisarão adaptar o trecho que vai da linha 400 até a linha 405. Consulte o artigo de *Programação BASIC* que trata da substituição da função **INKEY$** nesse microcomputador e faça, então, as alterações que forem necessárias.

## AS PERNAS DA ARANHA

```
500 LET temp=s(xpos)+s(xinc)
510 IF temp<1 OR temp>8+(3-pro
ps)*9 THEN  LET s(xinc)=-s(xin
c): GOTO 500
520 POKE 23607,60: PRINT AT s(
ypos),s(xpos);"  ";AT s(ypos)+
1,s(xpos);"  ": POKE 23607,252
530 LET s(ypos)=s(ypos)+s(yinc
): LET s(xpos)=temp: LET s(pic
ture)=1-s(picture): GOSUB 4200
540 RETURN
```

```
500 TE=S(XP)+S(XI)
510 IF TE<1 OR TE>8+(3-PP)*9 TH
EN S(XI)=-S(XI):GOTO 500
520 X2=S(XP)*8:Y2=S(YP)*8:PUT(X
2,Y2)-(X2+15,Y2+15),SP,PSET
530 S(YP)=S(YP)+S(YI):S(XP)=TE:
S(PI)=1-S(PI):GOSUB 4200
540 RETURN
```

```
500 TE=S(XP)+S(XI)
510 IF TE<1 OR TE>8+(3-PP)*9 TH
EN S(XI)=-S(XI):GOTO 500
530 S(YP)=S(YP)+S(YI):S(XP)=TE:
S(PI)=1-S(PI):GOSUB 4200
540 RETURN
```

```
500 TE = S(XP) + S(XI)
510  IF TE < 1 OR TE > 8 + (3 -
 PP) * 9 THEN S(XI) =  - S(XI):
 GOTO 500
520 X2 = S(XP) * 8:Y2 = S(YP) *
 8: HCOLOR= 0: DRAW S(PI) + 1 A
T X2,Y2: HCOLOR= 3
530 S(YP) = S(YP) + S(YI):S(XP)
 = TE:S(PI) = 1 - S(PI): GOSUB
4200
540  RETURN
```

A última rotina de animação trata da aranha marciana. Para tornar o jogo mais emocionante, ela não fica parada esperando por seu "almoço". Ao contrário, movimenta-se impacientemente entre a parede e as portas. Desenhamos, por isso, duas figuras para a aranha. O número da figura corrente, manipulado pela linha 530, é guardado em **s (picture)**, no programa do Spectrum, e em **S (PI)**, no dos demais micros.

As linhas 500 e 510 têm a função de impedir que a aranha escape de sua gaiola antes que todas as portas tenham sido removidas.

## ERA UMA VEZ UM BALÃO

```
600 PRINT AT b(ypos),b(xpos);
BRIGHT 1; INK b(colour);"gh";
AT b(ypos)+1,b(xpos);"ij"
610 POKE 23607,60
620 PRINT AT 0,14; INK 0;
PAPER 6;score
630 SOUND .5,-20
635 LET bl=bl-1: PRINT AT 0,7;
INK 0; PAPER 6;bl;: IF bl=9
THEN  PRINT  INK 0; PAPER 6;"
"
637 IF bl=0 THEN  LET bl=15+5*
level: LET level=level+1: LET
props=props-1: PRINT  INK 0;
PAPER 6;AT 0,7;bl;AT 0,2;level
: GOSUB 6000
640 PRINT AT b(ypos),b(xpos);"
  ";AT b(ypos)+1,b(xpos);"  "
650 PRINT AT ay,ax;"  ": LET a
x=29: LET ay=my+1
660 POKE 23607,252: GOSUB 4000
: GOSUB 5000: RETURN
```

```
600 X2=B(XP)*8:Y2=B(YP)*8:PUT (
X2,Y2)-(X2+15,Y2+15),GJ,PSET
620 COLOR 0:LINE(114,2)-(150,7)
,PSET,BF:NU=SC:DRAW"C1;BM114,2"
:GOSUB 1650
630 PLAY"V31;T200;O2;BAGFEDC;O1
;BAGFEDC"
635 BL=BL-1:COLOR0:LINE(58,2)-(
68,7),PSET,BF:DRAW"BM58,2;S2;C1
":NU=BL:GOSUB 1650
637 IF BL=0 THEN PP=3:LV=LV+1:B
```

```
L=15+5*LV:COLOR0:LINE(14,2)-(24
,7),PSET,BF:LINE(58,2)-(68,7),P
SET,BF:NU=LV:DRAW"BM14,2;C1":GO
SUB 1650:GOSUB 6000:NU=BL:DRAW"
BM58,2;C1":GOSUB 1650
640 X2=B(XP)*8:Y2=B(YP)*8:PUT(X
2,Y2)-(X2+15,Y2+15),SP,PSET
650 PUT(AX*8,AY*8)-(AX*8+15,AY*
8+7),S2,PSET:AX=29:AY=MY+1
660 GOSUB 4000:GOSUB 5000:RETUR
N
```

```
600 X2=B(XP)*8:Y2=B(YP)*8:PUT S
PRITE 4,(X2,Y2),9,BE
630 BL=BL-1
640 IF BL=0 THEN PP=3:LV=LV+1:B
L=15+5*LV
650 GOSUB 1700
660 PUT SPRITE 3,(AX*8,AY*8),4,
FL:AX=29:AY=MY
670 GOSUB 4000:GOSUB 5000:RETUR
N
```

```
600  HCOLOR= 3:X2 = B(XP) * 8:Y
2 = B(YP) * 8: DRAW BE AT X2,Y2

610 BL = BL - 1
620  IF BL = 0 THEN PP = 3:LV =
 LV + 1:BL = 15 + 5 * LV
630  GOSUB 1700
640  HCOLOR= 0:X2 = B(XP) * 8:Y
2 = B(YP) * 8: DRAW BE AT X2,Y2

650  DRAW FL AT AX * 8,AY * 8:A
X = 28:AY = MY + 1: HCOLOR= 3
660  GOSUB 4000: GOSUB 5000: RE
TURN
```

Essa rotina promove o estouro do balão quando ele é atingido pela flecha. Ela é muito simples: apenas coloca na tela a imagem do balão estourado, apagando-a em seguida.

Depois de cada estouro, atualiza-se o número de balões restantes e, se for necessário, também o nível de dificuldade. A flecha deve voltar, então, à posição de Freddy, para que ele possa estourar o próximo balão.

## CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR LINHA

| LINHA | FABRICANTE | MODELO |
|---|---|---|
| Apple II + | Appletronica | Thor 2010 |
| Apple II + | CCE | MC-4000 Exato |
| Apple II + | CPA | Absolutus |
| Apple II + | CPA | Polaris |
| Apple II + | Digitus | DGT-AP |
| Apple II + | Dismac | D-8100 |
| Apple II + | ENIAC | ENIAC II |
| Apple II + | Franklin | Franklin |
| Apple II + | Houston | Houston AP |
| Apple II + | Magnex | DM II |
| Apple II + | Maxitronica | MX-2001 |
| Apple II + | Maxitronica | MX-48 |
| Apple II + | Maxitronica | MX-64 |
| Apple II + | Maxitronica | Maxitronic I |
| Apple II + | Microcraft | Craf II Plus |
| Apple II + | Milmar | Apple II Plus |
| Apple II + | Milmar | Apple Master |
| Apple II + | Milmar | Apple Senior |
| Apple II + | Omega | MC-400 |
| Apple II + | Polymax | Maxxi |
| Apple II + | Polymax | Poly Plus |
| Apple II + | Spectrum | Microengenho I |
| Apple II + | Spectrum | Spectrum ed |
| Apple II + | Suporte | Venus II |
| Apple II + | Sycomig | SIC I |
| Apple II + | Unitron | AP II |
| Apple II + | Victor do Brasil | Elppa II Plus |
| Apple II + | Victor do Brasil | Elppa Jr. |
| Apple IIe | Microcraft | Craft IIe |
| Apple IIe | Microdigital | TK-3000 IIe |
| Apple IIe | Spectrum | Microengenho II |
| MSX | Gradiente | Expert GPC-1 |
| MSX | Sharp | Hotbit HB-8000 |
| Sinclair Spectrum | Microdigital | TK-90X |
| Sinclair Spectrum | Timex | Timex 2000 |
| Sinclair ZX-81 | Apply | Apply 300 |
| Sinclair ZX-81 | Engebras | AS-1000 |
| Sinclair ZX-81 | Filcres | NEZ-8000 |
| Sinclair ZX-81 | Microdigital | TK-82C |
| Sinclair ZX-81 | Microdigital | TK-83 |
| Sinclair ZX-81 | Microdigital | TK-85 |
| Sinclair ZX-81 | Prologica | CP-200 |
| Sinclair ZX-81 | Ritas | Ringo R-470 |
| Sinclair ZX-81 | Timex | Timex 1000 |
| Sinclair ZX-81 | Timex | Timex 1500 |
| TRS-80 Mod. I | Dismac | D-8000 |
| TRS-80 Mod. I | Dismac | D-8001/2 |
| TRS-80 Mod. I | LNW | LNW-80 |
| TRS-80 Mod. I | Video Genie | Video Genie I |
| TRS-80 Mod.III | Digitus | DGT-100 |
| TRS-80 Mod.III | Digitus | DGT-1000 |
| TRS-80 Mod.III | Kemitron | Naja 800 |
| TRS-80 Mod.III | Prologica | CP-300 |
| TRS-80 Mod.III | Prologica | CP-500 |
| TRS-80 Mod.III | Sysdata | Sysdata III |
| TRS-80 Mod.III | Sysdata | Sysdata Jr. |
| TRS-80 Mod.IV | Multix | MX-Compacto |
| TRS-80 Mod.IV | Sysdata | Sysdata IV |
| TRS-Color | Codimex | CS-6508 |
| TRS-Color | Dynacom | MX-1600 |
| TRS-Color | LZ | Color 64 |
| TRS-Color | Microdigital | TKS-800 |
| TRS-Color | Prologica | CP-400 |

## CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR NOME

| FABRICANTE | MODELO | PAÍS | LINHA |
|---|---|---|---|
| Appletronica | Thor 2010 | Brasil | Apple II + |
| Apply | Apply 300 | Brasil | Sinclair ZX-81 |
| CCE | MC-4000 Exato | Brasil | Apple II + |
| CPA | Absolutus | Brasil | Apple II + |
| CPA | Polaris | Brasil | Apple II + |
| Codimex | CS-6508 | Brasil | TRS-Color |
| Digitus | DGT-100 | Brasil | TRS-80 Mod.III |
| Digitus | DGT-1000 | Brasil | TRS-80 Mod.III |
| Digitus | DGT-AP | Brasil | Apple II + |
| Dismac | D-8000 | Brasil | TRS-80 Mod. I |
| Dismac | D-8001/2 | Brasil | TRS-80 Mod. I |
| Dismac | D-8100 | Brasil | Apple II + |
| Dynacom | MX-1600 | Brasil | TRS-Color |
| ENIAC | ENIAC II | Brasil | Apple II + |
| Engebras | AS-1000 | Brasil | Sinclair ZX-81 |
| Filcres | NEZ-8000 | Brasil | Sinclair ZX-81 |
| Franklin | Franklin | USA | Apple II + |
| Gradiente | Expert GPC1 | Brasil | MSX |
| Houston | Houston AP | Brasil | Apple II + |
| Kemitron | Naja 800 | Brasil | TRS-80 Mod.III |
| LNW | LNW-80 | USA | TRS-80 Mod. I |
| LZ | Color 64 | Brasil | TRS-Color |
| Magnex | DM II | Brasil | Apple II + |
| Maxitronica | MX-2001 | Brasil | Apple II + |
| Maxitronica | MX-48 | Brasil | Apple II + |
| Maxitronica | MX-64 | Brasil | Apple II + |
| Maxitronica | Maxitronic I | Brasil | Apple II + |
| Microcraft | Craft II Plus | Brasil | Apple II + |
| Microcraft | Caft IIe | Brasil | Apple IIe |
| Microdigital | TK-3000 IIe | Brasil | Apple IIe |
| Microdigital | TK-82C | Brasil | Sinclair ZX-81 |
| Microdigital | TK-83 | Brasil | Sinclair ZX-81 |
| Microdigital | TK-85 | Brasil | Sinclair ZX-81 |
| Microdigital | TK-90X | Brasil | Sinclair Spectrum |
| Microdigital | TKS-800 | Brasil | TRS-Color |
| Milmar | Apple II Plus | Brasil | Apple II + |
| Milmar | Apple Master | Brasil | Apple II + |
| Milmar | Apple Senior | Brasil | Apple II + |
| Multix | MX-Compacto | Brasil | TRS-80 Mod.IV |
| Omega | MC-400 | Brasil | Apple II + |
| Polymax | Maxxi | Brasil | Apple II + |
| Polymax | Poly Plus | Brasil | Apple II + |
| Prologica | CP-200 | Brasil | Sinclair ZX-81 |
| Prologica | CP-300 | Brasil | TRS-80 Mod.III |
| Prologica | CP-400 | Brasil | TRS-Color |
| Prologica | CP-500 | Brasil | TRS-80 Mod.III |
| Ritas | Ringo R-470 | Brasil | Sinclair ZX-81 |
| Sharp | Hotbit HB-8000 | Brasil | MSX |
| Spectrum | Microengenho I | Brasil | Apple II + |
| Spectrum | Microengenho II | Brasil | Apple IIe |
| Spectrum | Spectrum ed | Brasil | Apple II + |
| Suporte | Venus II | Brasil | Apple II + |
| Sycomig | SIC I | Brasil | Apple II + |
| Sysdata | Sysdata III | Brasil | TRS-80 Mod.III |
| Sysdata | Sysdata IV | Brasil | TRS-80 Mod.IV |
| Sysdata | Sysdata Jr. | Brasil | TRS-80 Mod.III |
| Timex | Timex 1000 | USA | Sinclair ZX-81 |
| Timex | Timex 1500 | USA | Sinclair ZX-81 |
| Timex | Timex 2000 | USA | Sinclair Spectrum |
| Unitron | AP II | Brasil | Apple II + |
| Victor do Brasil | Elppa II Plus | Brasil | Apple II + |
| Victor do Brasil | Elppa Jr. | Brasil | Apple II + |
| Video Genie | Video Genie I | USA | TRS-80 Mod. I |

## UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

**INPUT** foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:

Sinclair ZX-81

TRS-80

TK-2000

MSX

Spectrum

TRS-Color

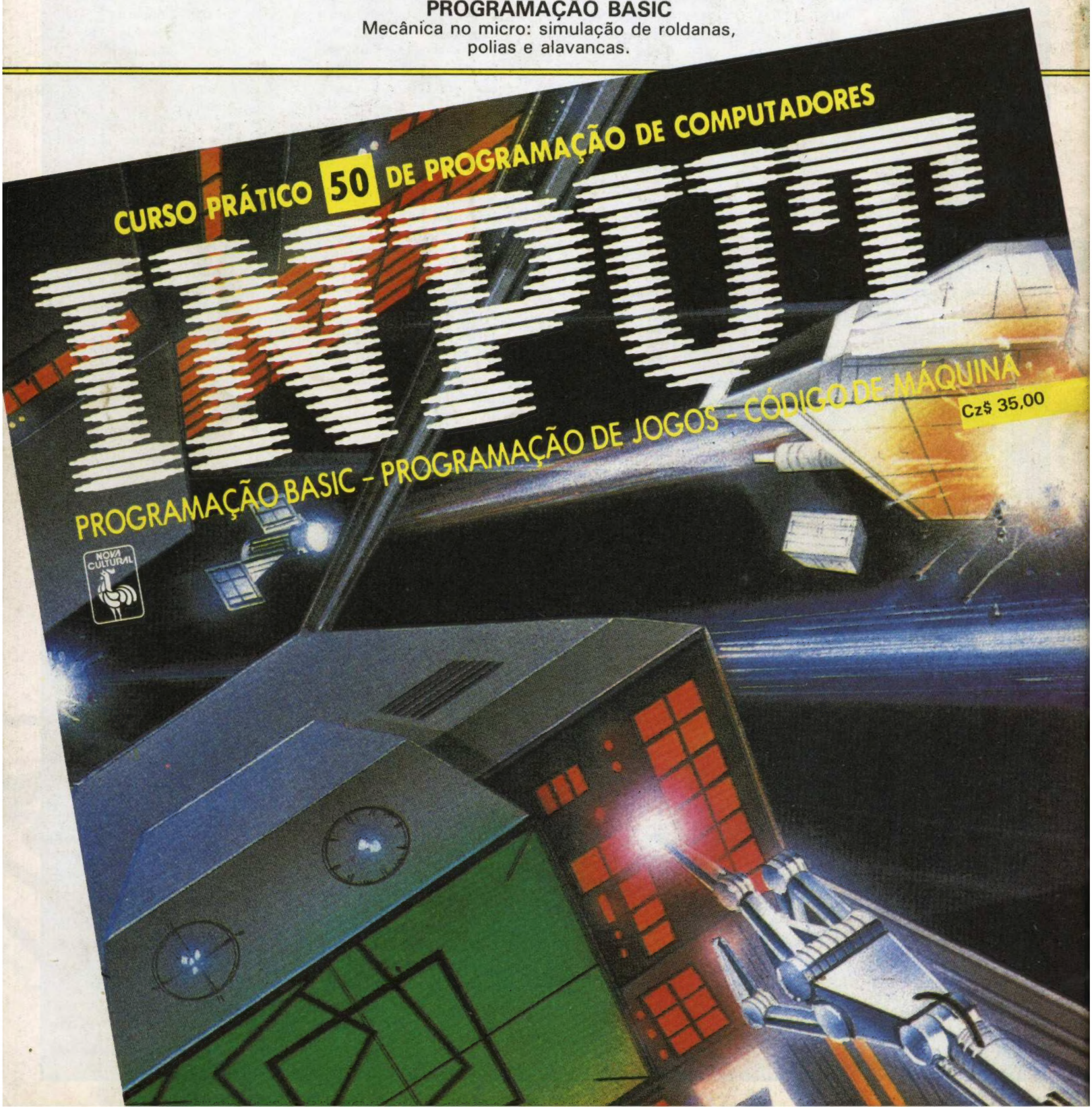
Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

# NO PRÓXIMO NÚMERO

**PROGRAMAÇÃO DE JOGOS**
Jogos de guerra no micro. Organização do jogo. Regras da batalha. Funcionamento da tela. Criação dos blocos gráficos.

**CÓDIGO DE MÁQUINA**
*Avalanche*: a sincronia dos diversos eventos. Rotinas encarregadas do acerto inicial das variáveis.

**PROGRAMAÇÃO BASIC**
Os diferentes métodos de controle de jogos por teclas múltiplas. A matriz do teclado. Um teclado a quatro mãos.

**PROGRAMAÇÃO BASIC**
Mecânica no micro: simulação de roldanas, polias e alavancas.